

# **IBM WebSphere V5.1 Performance, Scalability, and High Availability**

## **WebSphere Handbook Series**

Select a starting topology for your  
WebSphere Web site

Workload manage Web server,  
servlet, and EJB requests

Explore high availability  
and security options



Birgit Roehm  
Balazs Csepregi-Horvath  
Pingze Gao  
Thomas Hikade  
Miroslav Holecy  
Tom Hyland  
Namie Satoh  
Rohit Rana  
Hao Wang





International Technical Support Organization

**IBM WebSphere V5.1 Performance, Scalability, and  
High Availability  
WebSphere Handbook Series**

June 2004

**Note:** Before using this information and the product it supports, read the information in “Notices” on page xvii.

## **Second Edition (June 2004)**

This edition applies to Version 5, Release 1, Modification 0 of IBM WebSphere Application Server Network Deployment.

© Copyright International Business Machines Corporation 2003, 2004. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.



# Contents

<b>Notices</b> .....	xvii
Trademarks .....	xviii
<b>Preface</b> .....	xix
The team that wrote this redbook .....	xix
Become a published author .....	xxiii
Comments welcome .....	xxiii
<b>Summary of changes</b> .....	xxv
June 2004, Second Edition .....	xxv
<b>Part 1. Getting started</b> .....	1
<b>Chapter 1. Overview and key concepts</b> .....	3
1.1 Objectives .....	4
1.1.1 Scalability .....	5
1.1.2 Workload management .....	5
1.1.3 Availability .....	6
1.1.4 Maintainability .....	7
1.1.5 Session state .....	8
1.1.6 Performance impacts of WebSphere Application Server security .....	9
1.2 WebSphere Application Server architecture .....	10
1.2.1 WebSphere Application Server Network Deployment components .....	10
1.2.2 Web clients .....	12
1.2.3 Java clients .....	13
1.3 Workload management .....	13
1.3.1 Web server workload management .....	14
1.3.2 Plug-in workload management .....	14
1.3.3 Workload management using WebSphere clustering .....	16
1.3.4 Enterprise Java Services workload management .....	20
1.4 Managing session state among servers .....	21
1.4.1 HTTP sessions and the session management facility .....	22
1.4.2 EJB sessions or transactions .....	25
1.4.3 Server affinity .....	27
1.5 Performance improvements over previous versions .....	28
1.6 The structure of this redbook .....	29
<b>Chapter 2. Design for scalability</b> .....	33
2.1 Scaling your infrastructure .....	34

2.2	Understanding the application environment . . . . .	34
2.3	Categorizing your workload . . . . .	36
2.3.1	Workload patterns and Web site classifications . . . . .	36
2.3.2	Workload characteristics . . . . .	41
2.4	Determining the most affected components . . . . .	42
2.5	Selecting the scaling techniques to apply . . . . .	44
2.5.1	Using a faster machine . . . . .	45
2.5.2	Creating a cluster of machines . . . . .	46
2.5.3	Using appliance servers . . . . .	46
2.5.4	Segmenting the workload . . . . .	46
2.5.5	Batch requests . . . . .	47
2.5.6	Aggregating user data . . . . .	47
2.5.7	Managing connections . . . . .	48
2.5.8	Caching . . . . .	48
2.6	Applying the technique(s) . . . . .	49
2.7	Re-evaluating . . . . .	50
<b>Chapter 3.</b>	<b>Introduction to topologies . . . . .</b>	<b>53</b>
3.1	J2EE tiers model . . . . .	54
3.2	Topology selection criteria . . . . .	56
3.3	Strategies for scalability . . . . .	56
3.4	Single machine topology . . . . .	60
3.5	Separating the Web server . . . . .	61
3.6	Separating the database server . . . . .	65
3.7	Vertical scaling . . . . .	67
3.8	Horizontal scaling with clusters . . . . .	70
3.8.1	Horizontal scaling with IP sprayer . . . . .	71
3.9	One WebSphere administrative cell versus many . . . . .	74
3.10	Multiple clusters on one node versus one cluster per node . . . . .	76
3.11	The sample topology . . . . .	78
3.12	Topologies and high availability . . . . .	80
3.12.1	Using WebSphere Load Balancer custom advisor . . . . .	83
3.13	Closing thoughts on topologies . . . . .	86
3.14	Topology selection summary . . . . .	88
<b>Part 2.</b>	<b>Distributing the workload . . . . .</b>	<b>91</b>
<b>Chapter 4.</b>	<b>Web server load balancing . . . . .</b>	<b>93</b>
4.1	Introduction . . . . .	94
4.1.1	Scalability . . . . .	94
4.1.2	Availability . . . . .	94
4.1.3	Performance . . . . .	94
4.2	IBM WebSphere Edge Components . . . . .	95
4.3	Load Balancer overview . . . . .	96

4.3.1	Load Balancer topologies . . . . .	99
4.3.2	Installation and configuration . . . . .	103
4.3.3	Setting up the cluster machines . . . . .	103
4.3.4	Configuring a Web server cluster . . . . .	104
4.3.5	Testing the configuration . . . . .	107
4.4	Advisors . . . . .	110
4.4.1	Custom advisors . . . . .	112
4.4.2	Using WebSphere Application Server sample custom advisor . . . . .	113
4.5	Server affinity . . . . .	118
4.5.1	“Stickyness” to source IP address . . . . .	119
4.5.2	Passive cookie affinity . . . . .	119
4.5.3	Active cookie affinity . . . . .	120
4.5.4	URI . . . . .	120
4.5.5	SSL session ID . . . . .	120
4.6	Caching Proxy . . . . .	127
4.6.1	Forward proxy . . . . .	129
4.6.2	Reverse proxy (IP forwarding) . . . . .	129
4.6.3	Load Balancing . . . . .	130
4.6.4	Dynamic caching . . . . .	131
	<b>Chapter 5. Plug-in workload management and failover . . . . .</b>	<b>133</b>
5.1	Introduction . . . . .	134
5.2	The embedded HTTP transport . . . . .	135
5.3	Setting up the Web containers . . . . .	136
5.3.1	Virtual hosts . . . . .	136
5.3.2	Transports . . . . .	138
5.3.3	Creating clusters and cluster members . . . . .	143
5.4	WebSphere plug-in workload management . . . . .	144
5.4.1	Processing requests . . . . .	144
5.4.2	The plug-in configuration file . . . . .	147
5.4.3	Generation of the plug-in configuration file . . . . .	151
5.4.4	Plug-in workload management and failover policies . . . . .	156
5.5	Session management . . . . .	166
5.5.1	Session affinity . . . . .	167
5.5.2	Session failover inside the plug-in . . . . .	168
5.5.3	Session identifiers . . . . .	170
5.5.4	Session persistence and failover . . . . .	173
5.6	Troubleshooting the Web server plug-in . . . . .	180
5.6.1	Logging . . . . .	180
5.6.2	Trace . . . . .	183
5.7	Web server plug-in behavior and failover . . . . .	184
5.7.1	Normal operation . . . . .	185
5.7.2	Failover operation . . . . .	196

5.7.3 Tuning failover. . . . .	207
<b>Chapter 6. EJB workload management. . . . .</b>	<b>215</b>
6.1 Enabling EJB workload management . . . . .	216
6.2 EJB types and workload management . . . . .	217
6.2.1 Stateless session beans . . . . .	217
6.2.2 Stateful session beans . . . . .	218
6.2.3 Entity beans . . . . .	218
6.3 Naming and name spaces. . . . .	220
6.3.1 Looking up an EJB home with JNDI examples . . . . .	220
6.4 How EJBs participate in workload management . . . . .	228
6.4.1 Initial request. . . . .	228
6.4.2 Subsequent requests . . . . .	230
6.4.3 Cluster run state changes . . . . .	230
6.5 EJB server selection policy . . . . .	232
6.5.1 Server weighted round robin routing configuration . . . . .	234
6.5.2 Prefer local configuration . . . . .	237
6.6 EJB workload management behavior . . . . .	239
6.6.1 WLM behaviors using server weighted round robin . . . . .	240
6.6.2 Prefer local . . . . .	243
6.6.3 Process affinity . . . . .	246
6.6.4 Transaction affinity . . . . .	246
6.7 EJB workload management failover . . . . .	246
6.7.1 Exceptions triggering automatic failover . . . . .	247
6.7.2 Exceptions thrown by WLM to the application . . . . .	248
6.8 Backup Cluster support . . . . .	248
<b>Part 3. Implementing the solution . . . . .</b>	<b>253</b>
<b>Chapter 7. Implementing the sample topology. . . . .</b>	<b>255</b>
7.1 Overview . . . . .	256
7.1.1 Software products . . . . .	256
7.1.2 The sample topology. . . . .	256
7.1.3 Applications used in our sample topology. . . . .	259
7.2 Installation summary . . . . .	260
7.3 Configuring Caching Proxy . . . . .	260
7.3.1 Starting Caching Proxy . . . . .	261
7.3.2 Set up Caching Proxy . . . . .	261
7.3.3 Set administrator user ID and password . . . . .	262
7.3.4 Restart Caching Proxy . . . . .	263
7.4 Configuring Load Balancer . . . . .	264
7.4.1 Configuring the Web servers for Load Balancer . . . . .	265
7.4.2 Starting Load Balancer and administration GUI . . . . .	266
7.4.3 Connecting to the Dispatcher host . . . . .	266

7.4.4	Adding the Web server cluster . . . . .	267
7.4.5	Adding a port to the cluster . . . . .	270
7.4.6	Adding the Web servers to the cluster . . . . .	271
7.4.7	Start the Network Dispatcher manager . . . . .	273
7.4.8	Checking what you have done until now . . . . .	276
7.5	Configuring WebSphere clusters . . . . .	276
7.5.1	Introduction . . . . .	276
7.5.2	Creating the Web container cluster . . . . .	279
7.5.3	Creating the EJB cluster . . . . .	283
7.5.4	Configure persistent session management . . . . .	285
7.6	Installing and configuring BeenThere . . . . .	290
7.6.1	BeenThere installation summary . . . . .	290
7.6.2	Install BeenThere . . . . .	290
7.6.3	Regenerate Web server plug-in . . . . .	294
7.6.4	Restart servers . . . . .	296
7.6.5	Verifying BeenThere . . . . .	296
7.7	Installing and configuring Trade3.1 . . . . .	298
7.7.1	Trade3.1 installation summary . . . . .	299
7.7.2	Download the Trade3.1 package . . . . .	299
7.7.3	Set up and configure Trade3DB database . . . . .	299
7.7.4	Create JDBC and JMS resources . . . . .	301
7.7.5	SOAPify Trade3.ear . . . . .	302
7.7.6	Install Trade3.1 from the WebSphere Administrative Console . . . . .	302
7.7.7	Regenerate Web server plug-in . . . . .	306
7.7.8	Restart servers . . . . .	306
7.7.9	Install Trade3.1 using the installation script . . . . .	306
7.7.10	Working with Trade3.1 . . . . .	308
7.7.11	Verify failover with Trade3.1 . . . . .	311
7.7.12	Volume testing Trade3.1 . . . . .	311
<b>Part 4.</b>	<b>High availability solutions . . . . .</b>	<b>313</b>
<b>Chapter 8.</b>	<b>High availability concepts . . . . .</b>	<b>315</b>
8.1	Process availability and data availability . . . . .	316
8.2	Clustering for high availability . . . . .	316
8.3	Availability definition . . . . .	319
8.3.1	Levels of availability . . . . .	320
8.3.2	Availability matrix . . . . .	323
8.3.3	Causes of downtime . . . . .	325
8.3.4	Possible single points of failure in the WebSphere system . . . . .	326
8.3.5	Levels of WebSphere system availability . . . . .	328
8.3.6	Planning and evaluating your WebSphere HA solutions . . . . .	336
8.4	Failover terms and mechanisms . . . . .	337

<b>Chapter 9. WebSphere Application Server failover and recovery</b>	<b>345</b>
9.1 Overview	346
9.2 Web container clustering and failover	347
9.2.1 Web container failures and failover	348
9.2.2 Web server plug-in failover performance tuning	349
9.2.3 Network failures	354
9.2.4 Stream and overloading failover	356
9.3 HTTP session failover	357
9.3.1 Session affinity and failover	360
9.3.2 Session update methods and failover session data loss	362
9.3.3 Session persistence and failover	364
9.4 EJB container failover	365
9.4.1 EJB client redundancy and bootstrap failover support	366
9.4.2 EJB container redundancy and EJB WLM failover support	367
9.4.3 EJB WLM routing	368
9.4.4 LSD failover	371
9.4.5 EJB container failover behavior and tuning	371
9.4.6 Fault isolation and data integrity	374
9.4.7 EJB caching and failover	374
9.4.8 EJB types and failover	374
9.4.9 Conditions of WLM failover	375
9.4.10 Resource redundancy (EJB database, JMS resource, LDAP)	376
9.5 Enhancing WebSphere HA using clustering software	376
9.5.1 Failover unit	377
9.5.2 Configuration and setup for HACMP	378
9.5.3 Failover process	387
9.5.4 Advantages	387
 <b>Chapter 10. Deployment Manager and Node Agent high availability</b>	 <b>389</b>
10.1 Introduction	390
10.2 Node Agent failures	391
10.2.1 Application servers	392
10.2.2 Deployment Manager	394
10.2.3 Location Service Daemon	395
10.2.4 Naming server	395
10.2.5 Security server	396
10.2.6 Application clients	396
10.2.7 Synchronization Service and File Transfer Service	397
10.2.8 RAS service, PMI and monitoring	397
10.2.9 Administrative clients	397
10.3 Enhancing Node Agent high availability	398
10.3.1 Add Node Agent as OS daemon	399
10.3.2 Enhancing Node Agent HA using clustering software	399

10.4	Deployment Manager failures . . . . .	400
10.4.1	Configuration management . . . . .	401
10.4.2	Node Agent . . . . .	402
10.4.3	Application server . . . . .	403
10.4.4	Naming server . . . . .	403
10.4.5	Security server . . . . .	403
10.4.6	WLM runtime service . . . . .	404
10.4.7	Application clients . . . . .	404
10.4.8	Synchronization Service and File Transfer Service . . . . .	404
10.4.9	RAS Service and PMI monitoring . . . . .	404
10.4.10	Administrative clients . . . . .	404
10.5	Enhancing Deployment Manager high availability . . . . .	406
10.5.1	Add Deployment Manager to OS daemon service . . . . .	406
10.5.2	Make Deployment Manager highly available using clustering software and hardware . . . . .	406
 <b>Chapter 11. WebSphere Embedded JMS server and WebSphere MQ high availability . . . . .</b>		 417
11.1	JMS support in IBM WebSphere Application Server Network Deployment V5.1 . . . . .	418
11.2	Embedded JMS server high availability using clustering software . . . .	421
11.3	WebSphere MQ HA using clustering software . . . . .	424
11.4	WebSphere MQ built-in queue manager clustering . . . . .	425
11.5	Combined approaches for MQ built-in queue manager cluster and HACMP clustering . . . . .	430
 <b>Chapter 12. WebSphere data management high availability . . . . .</b>		 435
12.1	WebSphere with IBM HACMP . . . . .	436
12.1.1	Introduction and considerations . . . . .	436
12.1.2	Hardware and software . . . . .	440
12.1.3	Setup and configuration . . . . .	440
12.1.4	Typical failover . . . . .	444
12.1.5	Initiating various failures and tests . . . . .	446
12.1.6	Tuning heartbeat and cluster parameters . . . . .	447
12.2	WebSphere with MC/ServiceGuard . . . . .	448
12.2.1	Introduction and considerations . . . . .	449
12.2.2	Hardware and software . . . . .	450
12.2.3	Setup and configuration . . . . .	450
12.2.4	Typical failover . . . . .	454
12.2.5	Initiating various failures and tests . . . . .	456
12.2.6	Tuning heartbeat and cluster parameters . . . . .	458
12.3	WebSphere with Sun Cluster . . . . .	459
12.3.1	Introduction and considerations . . . . .	460

12.3.2 Setup and configuration . . . . .	461
12.4 WebSphere with VERITAS Cluster . . . . .	463
12.4.1 Introduction and considerations . . . . .	463
12.4.2 Setup and configuration . . . . .	464
12.5 WebSphere with Microsoft Cluster Service . . . . .	465
12.5.1 Introduction and considerations . . . . .	465
12.5.2 Setup and configuration . . . . .	466
12.6 Programming clients for transparent failover . . . . .	468
12.6.1 Bootstrap to multiple hosts . . . . .	468
12.6.2 Catch StaleConnectionException . . . . .	469
12.7 WebSphere and IP-based database failover . . . . .	475
12.7.1 Administrative servers and administrative actions . . . . .	475
12.7.2 Application servers . . . . .	476
12.7.3 Persistent session . . . . .	476
12.7.4 Java/C++ applications and application re-connecting . . . . .	476
12.7.5 Enterprise beans . . . . .	477
12.7.6 Web-based clients, applets, servlets, and JSPs . . . . .	477
12.7.7 Naming service and security service . . . . .	477
12.7.8 Workload management . . . . .	477
12.8 WebSphere with Oracle Parallel Server (OPS) and Real Application Cluster (RAC) . . . . .	478
12.8.1 Introduction and considerations . . . . .	478
12.8.2 Setup and configuration . . . . .	479
12.9 WebSphere with DB2 Parallel Server . . . . .	489
12.9.1 Introduction and considerations . . . . .	489
12.9.2 Setup and configuration . . . . .	490
12.10 WebSphere and non-IP-based database failover . . . . .	492
12.11 One instance for all WebSphere databases? . . . . .	493
12.12 WebSphere and Web server high availability . . . . .	498
<b>Chapter 13. High availability of LDAP, NFS, and firewall . . . . .</b>	<b>503</b>
13.1 Load Balancer high availability . . . . .	504
13.2 LDAP high availability . . . . .	507
13.3 Firewall high availability . . . . .	514
13.3.1 Using clustering software . . . . .	515
13.3.2 Using a network sprayer . . . . .	516
13.3.3 Conclusions . . . . .	520
13.4 Network file system high availability . . . . .	521
13.5 Summary . . . . .	522
<b>Part 5. Performance monitoring, tuning, and coding practices . . . . .</b>	<b>525</b>
<b>Chapter 14. Dynamic caching . . . . .</b>	<b>527</b>
14.1 Introduction . . . . .	528



14.1.1	WWW caching services	528
14.1.2	Fragment caching	531
14.1.3	Dynamic caching scenarios	533
14.2	Using WebSphere dynamic cache services	534
14.2.1	Installing Dynamic Cache Monitor	534
14.2.2	Enabling dynamic cache service	540
14.3	WebSphere dynamic caching scenarios	543
14.3.1	Servlet/JSP result caching	544
14.3.2	Command result caching	550
14.3.3	Cache replication	555
14.3.4	Cache invalidation	563
14.3.5	Troubleshooting the dynamic cache service	564
14.4	WebSphere external caching scenarios	566
14.4.1	WebSphere External Cache configuration	568
14.4.2	External caching by Web server plug-in	570
14.4.3	External caching on IBM HTTP Server	576
14.4.4	External caching on Caching Proxy	578
14.4.5	External cache invalidation	583
14.5	Conclusion	583
14.6	Benchmarking Trade3	585
14.6.1	Dynamic caching	585
14.6.2	Edge Side Includes	588
14.7	Reference	590
<b>Chapter 15.</b>	<b>Understanding and optimizing use of JMS components</b>	<b>591</b>
15.1	Introduction	592
15.2	Components used in a JMS configuration	592
15.2.1	Generic JMS usage	593
15.2.2	JMS and Message Driven Beans	596
15.3	Managing workload for JMS through configuration	599
15.3.1	Basic workload patterns	600
15.4	Relationships between JMS components	605
15.4.1	Component relationships when using MDBs	605
15.4.2	Component relationships when using generic JMS	608
15.5	Choosing optimal configuration settings	610
15.5.1	Creation of the JMS provider objects at the correct scope	611
15.5.2	Important JMS component settings	614
15.5.3	The listener service and listener ports	621
15.5.4	Setting up the QCF / TCF pools	628
15.5.5	More information	630
15.6	Optimizing JMS performance in the code	630
15.6.1	Selectors	630
15.6.2	Application defined persistence	635

15.6.3 Tidying up . . . . .	635
15.7 JMS Listener port failure behavior. . . . .	636
15.7.1 Failure in the listener port . . . . .	636
15.7.2 Failure to process a message . . . . .	638
15.8 Example JMS topologies and scenarios . . . . .	639
15.8.1 What does Trade3 use JMS for? . . . . .	641
15.8.2 Clustered Trade3 application and the Embedded JMS server . . .	642
15.8.3 Clustered Trade3 application WebSphere MQ and WebSphere Business Integration Event Broker . . . . .	651
15.9 Monitoring JMS performance within Tivoli Performance Viewer . . . . .	679
<b>Chapter 16. Server-side performance and analysis tools . . . . .</b>	<b>685</b>
16.1 The dimensions of monitoring . . . . .	686
16.1.1 Overview: Collecting and displaying application server data . . .	687
16.2 Performance Monitoring Infrastructure . . . . .	687
16.2.1 Performance data classification . . . . .	689
16.2.2 Performance data hierarchy . . . . .	690
16.2.3 Performance data counters. . . . .	695
16.2.4 Instrumentation levels . . . . .	696
16.2.5 Enabling the PMI service . . . . .	697
16.2.6 Setting instrumentation levels . . . . .	700
16.2.7 Using JVMPi facility for PMI statics. . . . .	701
16.2.8 Summary. . . . .	705
16.3 Using Tivoli Performance Viewer . . . . .	705
16.3.1 About Tivoli Performance Viewer . . . . .	705
16.3.2 What can Tivoli Performance Viewer do? . . . . .	707
16.3.3 Running Tivoli Performance Viewer . . . . .	708
16.3.4 Setting the instrumentation levels . . . . .	712
16.3.5 Using Tivoli Performance Viewer to monitor an application . . . .	716
16.3.6 Getting online help . . . . .	722
16.4 Other performance monitoring and management solutions . . . . .	722
16.5 Developing your own monitoring application. . . . .	723
16.6 PMI Request Metrics . . . . .	723
16.6.1 Enabling and configuring PMI Request Metrics . . . . .	725
16.6.2 PMI Request Metrics trace record format . . . . .	726
16.6.3 Filters . . . . .	728
16.6.4 Example: Generating trace records from Performance Monitoring Infrastructure Request Metrics . . . . .	729
16.7 Dynamic Cache Monitor . . . . .	729
16.8 Monitoring the IBM HTTP Server . . . . .	730
16.8.1 Configure your IBM HTTP Server . . . . .	730
16.8.2 Starting the Windows performance monitor . . . . .	730
16.8.3 Selecting performance data . . . . .	731

16.8.4 IBM HTTP Server status page .....	731
16.9 Log Analyzer .....	733
16.10 ThreadAnalyzer Technology preview .....	734
16.10.1 Download and installation of ThreadAnalyzer .....	734
16.10.2 Using ThreadAnalyzer .....	734
16.10.3 Automatic deadlock detection .....	741
16.10.4 ThreadAnalyzer usage example .....	743
16.10.5 Getting online help .....	745
16.11 Performance Advisors .....	745
16.11.1 Runtime Performance Advisor configuration settings .....	747
16.11.2 Advice configuration settings .....	748
16.11.3 Using the Runtime Performance Advisor .....	749
16.11.4 Runtime Performance Advisor output .....	751
16.11.5 TPV Advisor configuration .....	753
16.11.6 Using TPV Advisor .....	754
16.11.7 TPV Advisor output .....	757
16.12 Reference .....	758
<b>Chapter 17. Development-side performance and analysis tools .....</b>	<b>759</b>
17.1 Introduction .....	760
17.2 The Profiler (profiling tools) .....	760
17.2.1 Overview .....	761
17.2.2 IBM Agent Controller .....	762
17.2.3 Profiler configuration .....	764
17.2.4 Profiling the application .....	775
17.2.5 Profiler views .....	776
17.3 IBM Page Detailer .....	792
17.3.1 Overview .....	793
17.3.2 Important considerations .....	797
17.3.3 Key factors .....	798
17.3.4 Tips for using Page Detailer .....	799
17.3.5 Reference .....	802
<b>Chapter 18. Application development: Best practices for performance and scalability .....</b>	<b>803</b>
18.1 Introduction and general comments .....	804
18.2 Memory .....	805
18.3 Session Management .....	808
18.4 Synchronization .....	810
18.5 Servlets and JavaServer Pages .....	811
18.6 Logging .....	812
18.7 Enterprise JavaBeans .....	813
18.8 Database access .....	816

18.9 General coding issues . . . . .	817
18.10 Reference . . . . .	819
<b>Chapter 19. Performance tuning . . . . .</b>	<b>821</b>
19.1 Testing the performance of an application . . . . .	822
19.1.1 Introduction to application performance testing . . . . .	822
19.2 Tools of the trade . . . . .	823
19.2.1 WebSphere Performance Tools . . . . .	823
19.2.2 ApacheBench . . . . .	824
19.2.3 OpenSTA . . . . .	827
19.2.4 Other testing tools . . . . .	835
19.3 Performance monitoring guidelines . . . . .	835
19.3.1 Monitoring using Tivoli Performance Viewer and Advisors . . . . .	836
19.3.2 Performance analysis . . . . .	838
19.4 Performance tuning guidelines . . . . .	843
19.4.1 Tuning parameter hotlist . . . . .	844
19.4.2 Parameters to avoid failures . . . . .	845
19.4.3 Hardware and capacity settings . . . . .	845
19.4.4 Adjusting WebSphere Application Server system queues . . . . .	846
19.4.5 Application assembly performance checklist . . . . .	863
19.4.6 Java tuning . . . . .	870
19.4.7 Operating system tuning . . . . .	883
19.4.8 The Web server . . . . .	887
19.4.9 Dynamic Cache Service . . . . .	900
19.4.10 Security settings . . . . .	901
19.4.11 Tuning Secure Sockets Layer . . . . .	902
19.4.12 Object Request Broker (ORB) . . . . .	903
19.4.13 XML parser selection . . . . .	905
19.4.14 Transaction service settings - transaction log . . . . .	905
19.4.15 DB2 tuning . . . . .	906
19.4.16 Additional reference materials . . . . .	910
<b>Part 6. Appendixes . . . . .</b>	<b>913</b>
<b>Appendix A. Backup/recovery of Network Deployment configuration . . . . .</b>	<b>915</b>
Network Deployment configurations . . . . .	916
Backup methods . . . . .	917
Node failure scenarios . . . . .	918
Failure of the Deployment Manager node . . . . .	918
Failure of a WebSphere Application Server Base node . . . . .	918
Node recovery . . . . .	920
Recovery using filesystem backup and restore methods . . . . .	921
Recovery using backupConfig and restoreConfig . . . . .	923
Conclusion . . . . .	927

Additional reference material . . . . .	927
<b>Appendix B. Sample URL rewrite servlet . . . . .</b>	<b>929</b>
Setting up the servlet . . . . .	930
Source code . . . . .	930
Steps to install SessionSampleURLRewrite servlet . . . . .	931
Installing the urltest Web module . . . . .	932
Adding the servlet to an installed application . . . . .	933
<b>Appendix C. Additional material . . . . .</b>	<b>935</b>
Locating the Web material . . . . .	935
Using the Web material . . . . .	936
System requirements for downloading the Web material . . . . .	936
How to use the Web material . . . . .	936
<b>Related publications . . . . .</b>	<b>937</b>
IBM Redbooks . . . . .	937
Other publications . . . . .	937
Online resources . . . . .	938
How to get IBM Redbooks . . . . .	945
Help from IBM . . . . .	945
<b>Index . . . . .</b>	<b>947</b>



# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:  
*IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.*

*The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:* INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

## COPYRIGHT LICENSE:


This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

# Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

@server™  
alphaWorks®  
developerWorks®  
eServer™  
ibm.com®  
iSeries™  
pSeries®  
z/OS®  
zSeries®  
AFS®  
AIX 5L™

AIX®  
CICS®  
Domino®  
DB2®  
HACMP™  
Informix®  
IBM®  
IMS™  
Lotus®  
MQSeries®  
OS/400®

Purify®  
Rational Suite®  
Rational®  
Redbooks™  
Redbooks (logo) ™  
RS/6000®  
SecureWay®  
TestStudio®  
Tivoli®  
WebSphere®

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

OpenSTA is a registered trademark of CYRANO, Inc.

Apache is a trademark of The Apache Software Foundation.

Other company, product, and service names may be trademarks or service marks of others.



# Preface

This IBM® Redbook discusses various options for scaling applications based on IBM WebSphere Application Server Network Deployment V5.1. It explores how a basic WebSphere® configuration can be extended to provide more computing power by better exploiting the power of each machine and by using multiple machines. It examines a number of techniques:

- ▶ Using the IBM WebSphere Edge Components' Load Balancer to distribute load among multiple Web servers.
- ▶ Using the WebSphere Web server plug-in to distribute the load from one Web server to multiple application servers in a server cluster.
- ▶ Using the WebSphere EJB workload management facility to distribute load at the EJB level.
- ▶ Using dynamic caching techniques to improve the performance of a Web site.
- ▶ Using clustering solutions such as HACMP™ to meet the high availability needs of critical applications.
- ▶ Using application development best practices to develop a scalable application.
- ▶ Using the performance tuning options available with WebSphere to adjust the application server configuration to the needs of your application.

This book provides step-by-step instructions for implementing a sample, multiple-machine environment. We use this environment to illustrate most of the IBM WebSphere Application Server Network Deployment V5.1 workload management and scalability features.

## The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.



*The team that wrote this redbook (from left to right): Thomas Hikade, Miroslav Holec, Namie Satoh, Tom Hyland, Birgit Roehm, Balazs Csepregi-Horvath, Pingze Gao. Pictured below: Rohit Rana and Hao Wang*

**Birgit Roehm** is a Project Leader at the International Technical Support Organization, Raleigh Center. She writes about various aspects of WebSphere and Domino®. Before joining the ITSO in 2003, Birgit worked in iSeries™ Advanced Technical Support, Germany, responsible for Domino and WebSphere on iSeries.

**Balazs Csepregi-Horvath** is a Customer Engineer working for IBM Global Services in Hungary. He holds a bachelor's degree in Heavy Current Automation and a bachelor's degree in Information Technology, both from Kando Kalman Technical College, Budapest, Hungary. He has four years of expertise supporting WebSphere and DB2® products and is an IBM Certified Advanced Technical Expert for IBM @server™ pSeries® and AIX® 5L™.

**Pingze Gao** is an IT Consultant at Integrated Opensource (an IBM Business Partner) in Austin, Texas. His more than 10 year IT experience spans from e-business infrastructure, design, testing, and implementation, to deployment of full life cycle software development. His areas of expertise include J2EE, security, performance/tuning, and integration with WebSphere Application Server, MQSeries/Workflow, and Portal Server. He is a Sun certified Java™ Architect and a strong proponent of open source, open standards, service oriented architecture, Web Services, best practices, design patterns, and Agile software development.

**Thomas Hikade** is an IT Specialist working for IBM Global Services Strategic Outsourcing Service Delivery in Austria. He has 10 years of experience in IT, of which he spent the last four years focusing on e-business infrastructure design, implementation, and operation. Thomas is a Red Hat Certified Engineer and holds a master's degree in Computer Science from the Technical University of Vienna, Austria. His main areas of expertise are problem determination, high availability and performance tuning of the WebSphere Application Server product family (WebSphere Application Server, WebSphere Commerce, WebSphere Portal, WebSphere MQ), but he is also experienced in implementing Internet services using various open source projects and tools.

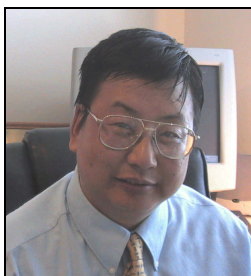
**Miroslav Holecy** is an Advisory IT Specialist working for IBM Global Services in Sweden. He has extensive experience with the architecture and the implementation of e-business solutions, with a strong focus on infrastructure and performance aspects. Miroslav is the co-author of two WebSphere Commerce Redbooks (SG24-6855, SG24-6180). His formal education is in Computer Science and he holds a master's degree in Artificial Intelligence studies from the Technical University of Kosice, Slovakia.

**Tom Hyland** is a WebSphere Specialist in the UK, working for the WebSphere Software services group in Warwick. He has nine years of experience in IT, specializing in the WebSphere and Java fields for the last five years. He holds a first class bachelor's degree in Computer Science from IBM and Portsmouth University. His main area of expertise is consulting on the WebSphere foundation and tools products, in particular, performance, security, and availability of WebSphere Application Server in production environments.

**Namie Satoh** is an IT Specialist at IBM Systems Engineering Co. Ltd (ISE), part of the ATS function in Japan. She has three years of experience in the Java and WebSphere fields and provides technical support in WebSphere. She holds a degree in Literature from Ochanomizu University in Tokyo, Japan. Lately she has been engaged mainly in Web Services and Autonomic Computing.



**Rohit Rana** is a Software Engineer working for IBM Global Services in India. He has five years of experience in e-business. For the last four years, he has been porting applications to WebSphere. He holds a degree in Electrical Engineering from Jamia Millia, New Delhi, India and an MBA from IIFT, New Delhi, India. His main areas of expertise include WebSphere, Java, and Web architectures.



**Hao Wang** is a member of the WebSphere Development Team. He has been working on the San Francisco project (business Java shared framework), WebSphere solutions integration, performance and customer scenarios, WebSphere connection management, and WebSphere WLM/high availability. His background includes a Ph.D. degree in Computer Science from Iowa State University. Before he joined IBM in January 1999, he worked for the university and national laboratory as an Associate Professor and Scientist for more than 10 years, taught graduate-level courses in fields, such as the principles of database systems, and conducted research and development on high performance distributed and parallel computing, cluster computing, and computer simulation models. He also worked as an IT Consultant to help many of the biggest companies in the world. He has published dozens of papers in world-class journals, conferences, and books.

Thanks to the following people for their contributions to this project:

Mark Endrei, Peter Kovari, Linda Robinson, Carla Sadtler, Margaret Ticknor,  
Jeanne Tucker  
*International Technical Support Organization, Raleigh Center*

Denise Barlock, Ondrej Bizub, Srinivas Hasti, Matt Hogstrom, Ken Hygh,  
Priyanka Jain, Stacy Joines, Bert Laonipon, Jakob L. Mickley, Srin Ranganaswamy,  
Sree Ratnasinghe, Andrew Spyker, Rengan Sundararaman, Sharon Weed  
*IBM Raleigh*

Randall Baartman, Douglas Berg, Joe Bockhold, Mark Bransford, Erik  
Daughtrey, Surya Duggirala, Anthony Tuel, Rob Wisniewski  
*IBM Rochester*

LeRoy Krueger  
*IBM Atlanta*

Tom Alcott  
*IBM Costa Mesa*

John P. Cammarata  
*IBM Durham*

Bill Hines  
*IBM Mechanicsburg*

Eugene Chan, Stanley Shiah, Paul Slauenwhite  
*IBM Toronto*

Gerhard Poul  
*IBM Austria*

Denis Ley  
*IBM Germany*

Maurits JA Andre  
*IBM Netherlands*

Drago Kralj  
*IBM Slovenia*

Lukas Theiler  
*IBM Switzerland*

Ann Barrago, Marc Carter, David Currie, Steve Cox, Paul Gover, Andrew Leonard, Jonathan Marshall, Ian Parkinson, Andrew Piper, Philip Thomas, Andrew J. Wasley  
*IBM United Kingdom*

## Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- Send your comments in an Internet note to:

[redbook@us.ibm.com](mailto:redbook@us.ibm.com)

- Mail your comments to:

IBM Corporation, International Technical Support Organization  
Dept. HZ8 Building 662  
P.O. Box 12195  
Research Triangle Park, NC 27709-2195

# Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes

for SG24-6198-01

for *IBM WebSphere V5.1 Performance, Scalability, and High Availability:*  
*WebSphere Handbook Series*

as created or updated on June 10, 2004.

## June 2004, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described below.

### New information

- ▶ New Chapter 14, “Dynamic caching” on page 527
- ▶ New Chapter 15, “Understanding and optimizing use of JMS components” on page 591
- ▶ New Appendix A, “Backup/recovery of Network Deployment configuration” on page 915

### Changed information

- ▶ Minor changes/revisions in Chapters 1, 2, 3, and 4.
- ▶ Chapter 5, “Plug-in workload management and failover” on page 133
- ▶ Chapter 6, “EJB workload management” on page 215
- ▶ Chapter 7, “Implementing the sample topology” on page 255
- ▶ Chapter 9, “WebSphere Application Server failover and recovery” on page 345
- ▶ Chapter 10, “Deployment Manager and Node Agent high availability” on page 389
- ▶ Chapter 16, “Server-side performance and analysis tools” on page 685
- ▶ Chapter 17, “Development-side performance and analysis tools” on page 759

- ▶ Chapter 19, “Performance tuning” on page 821

### **Removed information**

- ▶ Security introduction (was Chapter 8 in the previous version of the redbook). Refer to *IBM WebSphere V5.0 Security*, SG24-6573, for detailed information about WebSphere security.





# Part 1

# Getting started





# Overview and key concepts

This chapter provides a conceptual overview of the goals and issues associated with scaling applications in the WebSphere environment. It also presents the various techniques that can be used to implement scaling.

A short section highlights the performance improvements of WebSphere Application Server V5.1 over previous V5.x versions (see “Performance improvements over previous versions” on page 28).

A prerequisite for all WebSphere scalability and workload management concepts discussed in this redbook is the use of IBM WebSphere Application Server Network Deployment V5.1 or IBM WebSphere Application Server Enterprise V5.1. We assume the use of IBM WebSphere Application Server Network Deployment throughout this book.

## 1.1 Objectives

As outlined by the product documentation, a typical minimal configuration for a WebSphere Application Server V5.1 installation is illustrated in Figure 1-1. There is a single Web (HTTP) server and a single application server, and both are co-resident on a single machine. Obviously, the performance characteristics of this setup are limited by the power of the machine on which it is running, and by various constraints inherent in the configuration itself and in the implementation of WebSphere Application Server. A short introduction to the various components of the WebSphere runtime is provided in 1.2, “WebSphere Application Server architecture” on page 10.

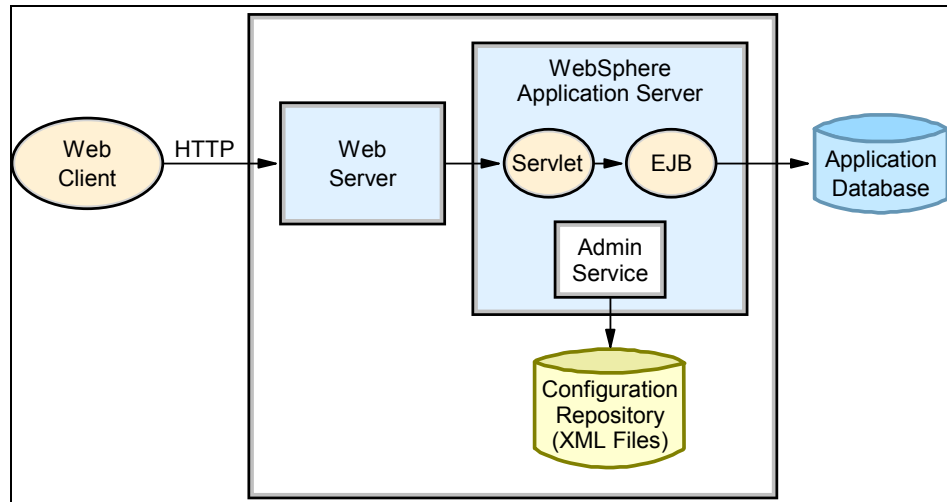


Figure 1-1 Basic WebSphere configuration

The objective of this book is to explore how this basic configuration can be extended to provide more computing power, by better exploiting the power of each machine, and by using multiple machines. Specifically, we are interested in defining system configurations that exhibit the following properties:

- ▶ Scalability
- ▶ Workload management
- ▶ Availability
- ▶ Maintainability
- ▶ Session management
- ▶ Performance impacts of WebSphere Application Server security

Note that scaling the application server environment does not help if your application has an unscalable design. For Web application and EJB development best practices, refer to Chapter 18, “Application development: Best practices for performance and scalability” on page 803 and to the white paper *WebSphere Application Server Development Best Practices for Performance and Scalability*, found at:

[http://www.ibm.com/software/webservers/appserv/ws\\_bestpractices.pdf](http://www.ibm.com/software/webservers/appserv/ws_bestpractices.pdf)

The IBM developerWorks® WebSphere Best Practices Zone provides additional information. Refer to the following Web sites:

<http://www.ibm.com/developerworks/websphere/zones/bp/background.html>

<http://www.ibm.com/developerworks/websphere/zones/bp/index.html>

### 1.1.1 Scalability

The proposed configurations should allow the overall system to service a higher client load than that provided by the simple basic configuration. Ideally, it should be possible to service any given load simply by adding the appropriate number of servers or machines.

#### Performance

Performance involves minimizing the response time for a given transaction load.

#### Throughput

Throughput, while related to performance, more precisely defines the number of concurrent transactions that can be accommodated.

### 1.1.2 Workload management

The proposed configurations should ensure that each machine or server in the configuration processes a fair share of the overall client load that is being processed by the system as a whole. In other words, it is not efficient to have one machine overloaded while another machine is mostly idle. If all machines have roughly the same power, each should process a roughly equal share of the load. Otherwise the workload should be based on weights associated with each machines processing power assuming no affinity configuration.

Furthermore, if the total load changes over time, the system should automatically adapt itself, for example, all machines may use 50% of their capacity, or all machines may use 100% of their capacity. But not one uses 100% of its capacity and rest uses 15% of their capacity.

In this redbook, we discuss Web server load balancing using both WebSphere Edge Components and WebSphere workload management techniques.

### 1.1.3 Availability

Availability requires that the topology provide some degree of process redundancy in order to eliminate single points of failure. While vertical scalability can provide this by creating multiple processes, the physical machine then becomes a single point of failure. For this reason, a high availability topology typically involves horizontal scaling across multiple machines.

#### **Hardware-based high availability**

Using a WebSphere Application Server multiple machine configuration eliminates a given application server process as a single point of failure. In WebSphere Application Server V5.0 and higher, the removal of the dependency on the administrative server process further reduces the potential that a single process failure can disrupt processing on a given node. In fact, the only single point of failure in a WebSphere cell/cluster is the Network Deployment Manager where all central administration is done. However, a failure at the Deployment Manager is less catastrophic. Application servers are more self sufficient in WebSphere Application Server V5.0 and higher compared to WebSphere Application Server V4.x. To provide high availability, an external High Availability solution should be examined for the Deployment Manager of WebSphere Application Server V5.0 and higher.

See Chapter 10, “Deployment Manager and Node Agent high availability” on page 389 for further details.

#### **Failover**

The proposition to have multiple servers (potentially on multiple independent machines) naturally leads to the potential for the system to provide failover. That is, if any one machine or server in the system were to fail for any reason, the system should continue to operate with the remaining servers. The load-balancing property should ensure that the client load gets redistributed to the remaining servers, each of which will take on a proportionately slightly higher percentage of the total load. Of course, such an arrangement assumes that the system is designed with some degree of overcapacity, so that the remaining servers are indeed sufficient to process the total expected client load.

Ideally, the failover aspect should be totally transparent to clients of the system. When a server fails, any client that is currently interacting with that server should be automatically redirected to one of the remaining servers, without any interruption of service and without requiring any special action on the part of that client. In practice, however, most failover solutions may not be completely

transparent. For example, a client that is currently in the middle of an operation when a server fails may receive an error from that operation, and may be required to retry (at which point the client would be connected to another, still-available server). Or the client may observe a “hiccup” or delay in processing, before the processing of its requests resumes automatically with a different server. The important point in failover is that each client, and the set of clients as a whole, is able to eventually continue to take advantage of the system and receive useful service, even if some of the servers fail and become unavailable. Conversely, when a previously-failed server is repaired and again becomes available, the system may transparently start using that server again to process a portion of the total client load.

The failover aspect is also sometimes called fault tolerance, in that it allows the system to survive a variety of failures or faults. It should be noted, however, that failover is only one technique in the much broader field of fault tolerance, and that no such technique can make a system 100 percent safe against every possible failure. The goal is to greatly minimize the *probability* of system failure, but keep in mind that the *possibility* of system failure cannot be completely eliminated.

Note that in the context of discussions on failover, the term *server* most often refers to a physical machine (which is typically the type of component that fails). But we will see that WebSphere also allows for the possibility of one server process on a given machine to fail independently, while other processes on that same machine continue to operate normally.

## 1.1.4 Maintainability

While maintainability is somewhat related to availability, there are specific issues that need to be considered when deploying a topology that is maintainable. In fact, some maintainability factors are at cross purposes to availability. For instance, ease of maintainability would dictate that one should minimize the number of application server instances in order to facilitate online software upgrades. Taken to the extreme, this would result in a single application server instance, which of course would not provide a high availability solution. In many cases, it is also possible that a single application server instance would not provide the required throughput or performance.

Some of the maintainability aspects that we consider are:

- ▶ Dynamic changes to configuration
- ▶ Mixed configuration
- ▶ Fault isolation

## **Dynamic changes to configuration**

In certain configurations, it may be possible to modify the configuration on the fly without interrupting the operation of the system and its service to clients. For example, it may be possible to add or remove servers to adjust to variations in the total client load. Or it may be possible to temporarily stop one server to change some operational or tuning parameters, then restart it and continue to serve client requests. Such characteristics, when possible, are highly desirable, since they enhance the overall manageability and flexibility of the system.

## **Mixed configuration**

In some configurations, it may be possible to mix multiple versions of a server or application, so as to provide for staged deployment and a smooth upgrade of the overall system from one software or hardware version to another. Coupled with the ability to make dynamic changes to the configuration, this property may be used to effect upgrades without any interruption of service.

## **Fault isolation**

In the simplest application of failover, we are only concerned with clean failures of an individual server, in which a server simply ceases to function completely, but this failure has no effect on the health of other servers. However, there are sometimes situations where one malfunctioning server may in turn create problems for the operation of other, otherwise healthy servers. For example, one malfunctioning server may hoard system resources, or database resources, or hold critical shared objects for extended periods of time, and prevent other servers from getting their fair share of access to these resources or objects. In this context, some configurations may provide a degree of fault isolation, in that they reduce the potential for the failure of one server to affect other servers.

### **1.1.5 Session state**

Unless you have only a single application server or your application is completely stateless, maintaining state between HTTP client requests will also play a factor in determining your configuration. In WebSphere V5.1 there are two methods for sharing of sessions between multiple application server processes (cluster members). One method is to persist the session to a database. An alternate approach is to use memory-to-memory session replication functionality, which was added to WebSphere V5 and is implemented using WebSphere internal messaging. The memory-to-memory replication (sometimes also referred to as “in-memory replication”) eliminates a single point of failure found in the session database (if the database itself has not been made highly available using clustering software). See “HTTP sessions and the session management facility” on page 22 for additional information.



**Tip:** Refer to the redbook *A Practical Guide to DB2 UDB Data Replication V8*, SG24-6828, for details about DB2 replication.

### 1.1.6 Performance impacts of WebSphere Application Server security

The potential to distribute the processing responsibilities between multiple servers and, in particular, multiple machines, also introduces a number of opportunities for meeting special security constraints in the system. Different servers or types of servers may be assigned to manipulate different classes of data or perform different types of operations. The interactions between the various servers may be controlled, for example through the use of firewalls, to prevent undesired accesses to data.

From the performance point of view, there are a few things to consider when designing a secure solution. Building up SSL communication causes extra HTTP requests and responses between the machines and every SSL message is encrypted on one side and decrypted on the other side. The authorization process adds additional load to the application server. In a distributed environment, the authorization server should be put onto a separate machine in order to offload application processing. The following three settings can help to fine-tune the security-related configurations to enhance performance:

- ▶ Security cache timeout

Its setting determines how long WebSphere should cache information related to permission and security credentials. When the cache timeout expires, all cached information becomes invalid. Subsequent requests for the information result in a database lookup. Sometimes, acquiring the information requires invoking an LDAP-bind or native authentication, both of which are relatively costly operations in terms of performance.

- ▶ HTTP session timeout

This parameter specifies how long a session will be considered active when it is unused. After the timeout, the session expires and another session object will be created. With high-volume Web sites, this may influence the performance of the server.

- ▶ Registry and database performance

Databases and registries used by an application influence the WebSphere Application Server performance. Especially in a distributed environment, when the authorization process uses an LDAP server, you have to consider tuning the LDAP database and LDAP server for performance first, before starting to tune WebSphere.

**Note:** WebSphere security is out of the scope of this redbook. However, an entire redbook is dedicated to this topic. See *IBM WebSphere V5.0 Security*, SG24-6573 for details.

## 1.2 WebSphere Application Server architecture

This section introduces the major components within WebSphere Application Server. Refer to the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195 for details about the WebSphere Application Server architecture and components.

### 1.2.1 WebSphere Application Server Network Deployment components

The following is a short introduction for each WebSphere Network Deployment runtime component and their functions:

- ▶ **Deployment Manager**

The Deployment Manager is part of an WebSphere Application Server Network Deployment V5.x installation. It provides a central point of administrative control for all elements in a distributed WebSphere cell. Deployment Manager is a required component for any vertical or horizontal scaling and there exists only one specimen of it at a time.

- ▶ **Node Agent**

The Node Agent communicates directly with the Deployment Manager, and is used for configuration synchronization and administrative tasks such as stopping/starting of individual application servers and performance monitoring on the application server node.

- ▶ **Application server (instance)**

An application server in WebSphere is the process that is used to run your servlet and/or EJB-based applications, providing both Web container and EJB container.

- ▶ **Web server and Web server plug-in**

While the Web server is not strictly part of the WebSphere runtime, WebSphere communicates with your Web server of choice: IBM HTTP Server powered by Apache, Microsoft® Internet Information Server, Apache, Sun ONE Web server, and Lotus® Domino via a plug-in. This plug-in communicates requests from the Web server to the WebSphere runtime.

- ▶ Embedded HTTP transport

The embedded HTTP transport is a service of the Web container. An HTTP client connects to a Web server and the HTTP plug-in forwards the requests to the embedded HTTP transport. The communication type is either HTTP or HTTPS between the plug-in and the embedded HTTP transport.

Although the embedded HTTP transport is available in any WebSphere Application Server to allow testing or development of the application functionality prior to deployment, it should *not* be used for production environments.

- ▶ Embedded Messaging (JMS) Server

The JMS server was introduced with WebSphere Application Server V5 (all versions except WebSphere Express). The JMS server is a complete messaging server that supports point-to-point and publish/subscribe styles of messaging.

- ▶ Administrative service

The administrative service runs within each WebSphere Application Server Java virtual machine (JVM.) Depending on the version installed, there can be administrative services installed in many locations. In WebSphere Application Server V5.0 and higher, the administrative service runs in each application server. In a Network Deployment configuration, the Deployment Manager, Node Agent, application server(s), and JMS server all host an administrative service. This service provides the ability to update configuration data for the application server and its related components.

- ▶ Administrative Console

The WebSphere Administrative Console provides an easy-to-use, graphical “window” to a WebSphere cell, runs in a browser, and connects directly to a WebSphere V5.1 server or to a Deployment Manager node that manages all nodes in a cell.

- ▶ Configuration repository

The configuration repository stores the configuration for all WebSphere Application Server components inside a WebSphere cell. The Deployment Manager communicates changes in configuration and runtime state to all cell members through the configuration repository.

- ▶ Administrative cell

A cell is a set of Application Servers managed by the same Deployment Manager. Some application server instances may reside on the same node, others on different nodes. Cell members not necessarily run the same application.

► Server cluster

A server cluster is a set of application server processes. They run the same application but usually on different nodes. The main purpose of clusters is load balancing and failover.

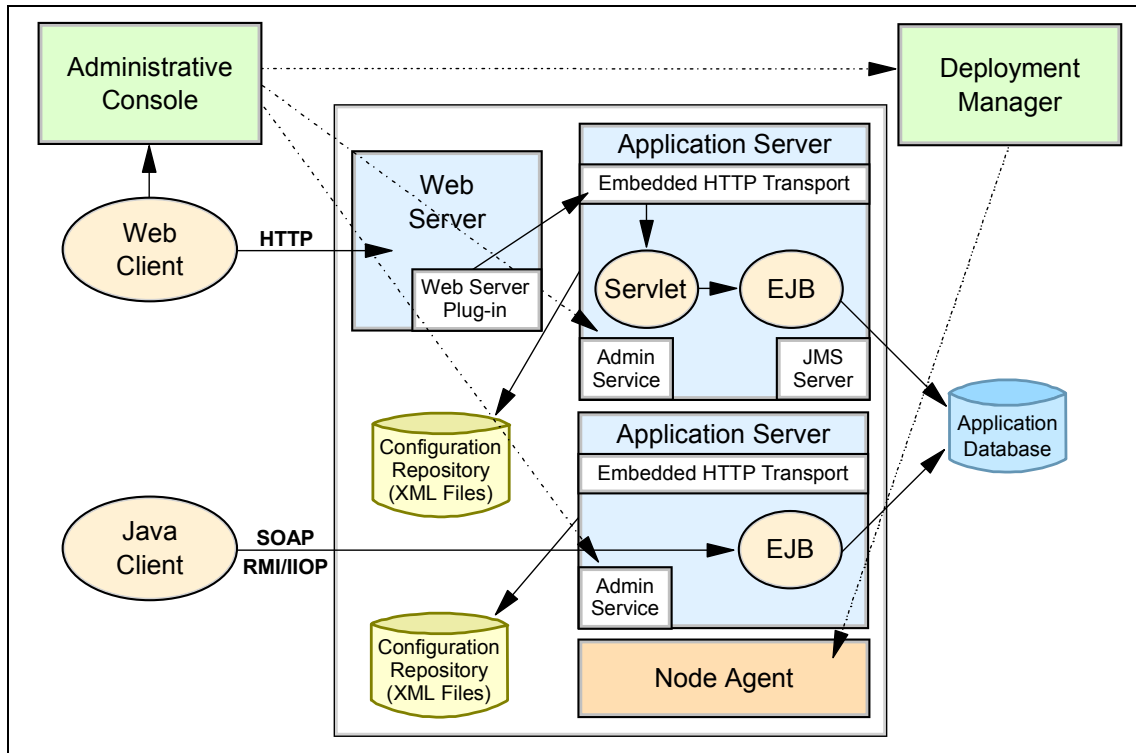


Figure 1-2 WebSphere configuration with Java clients and Web clients

## 1.2.2 Web clients

The basic configuration shown in Figure 1-1 on page 4 refers to a particular type of client for WebSphere characterized as a Web client. Such a client uses a Web browser to interact with the WebSphere Application Servers, through the intermediary of a Web server (and plug-in), which in turn invokes a servlet (you can also access static HTML pages) within WebSphere.

### 1.2.3 Java clients

Although Web clients constitute the majority of users of WebSphere today, WebSphere also supports another type of client characterized as a Java client. Such a client is a stand-alone Java program (either GUI-based or not), which uses the Java RMI/IIOP facilities to make direct method invocations on various EJB objects within an application server, without going through an intervening Web server and servlet, as shown in Figure 1-2 on page 12.

## 1.3 Workload management

Workload Management (WLM) is the process of spreading multiple requests for work over the resources that can do the work. It optimizes the distribution of processing tasks in the WebSphere Application Server environment. Incoming work requests are distributed to the application servers and other objects that can most effectively process the requests.

Workload management is also a procedure for improving performance, scalability, and reliability of an application. It provides failover when servers are not available.

Workload management is most effective when used in systems that contain servers on multiple machines. It can also be used in systems that contain multiple servers on a single, high-capacity machine. In either case, it enables the system to make the most effective use of the available computing resources.

Workload management provides the following benefits to WebSphere applications:

- ▶ It balances client processing requests, allowing incoming work requests to be distributed according to a configured WLM selection policy.
- ▶ It provides failover capability by redirecting client requests to a running server when one or more servers are unavailable. This improves the availability of applications and administrative services.
- ▶ It enables systems to be scaled up to serve a higher client load than provided by the basic configuration. With clusters and cluster members, additional instances of servers can easily be added to the configuration. See 1.3.3, “Workload management using WebSphere clustering” on page 16 for details.
- ▶ It enables servers to be transparently maintained and upgraded while applications remain available for users.
- ▶ It centralizes administration of application servers and other objects.

Two types of requests can be workload managed in IBM WebSphere Application Server Network Deployment V5.1:

- ▶ **HTTP requests** can be distributed across multiple Web containers, as described in 1.3.2, “Plug-in workload management” on page 14. We will refer to this as *Plug-in WLM*.
- ▶ **EJB requests** can be distributed across multiple EJB containers, as described in 1.3.4, “Enterprise Java Services workload management” on page 20. We will refer to this as *EJS WLM*.

### 1.3.1 Web server workload management

If your environment uses multiple Web servers, then a mechanism is needed to allow servers to share the load. The HTTP traffic must be spread among a group of servers. These servers must appear as one server to the Web client (browser), making up a *cluster*, which is a group of independent nodes interconnected and working together as a single system. This cluster concept should not be confused with a WebSphere cluster as described in 1.3.3, “Workload management using WebSphere clustering” on page 16.

As shown in Figure 1-3, a load balancing mechanism called *IP spraying* can be used to intercept the HTTP requests and redirect them to the appropriate machine on the cluster, providing scalability, load balancing, and failover.

See Chapter 4, “Web server load balancing” on page 93 for details.

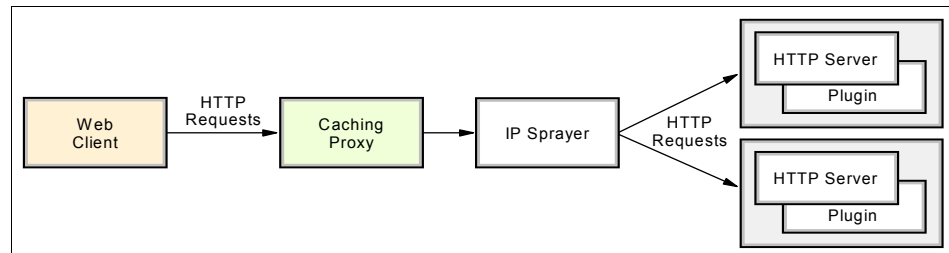


Figure 1-3 Web server workload management

### 1.3.2 Plug-in workload management

This is a short introduction into plug-in workload management, where servlet requests are distributed to the Web container in clustered application servers, as shown in Figure 1-4 on page 15. This configuration is also referred to as the *servlet clustering architecture*.

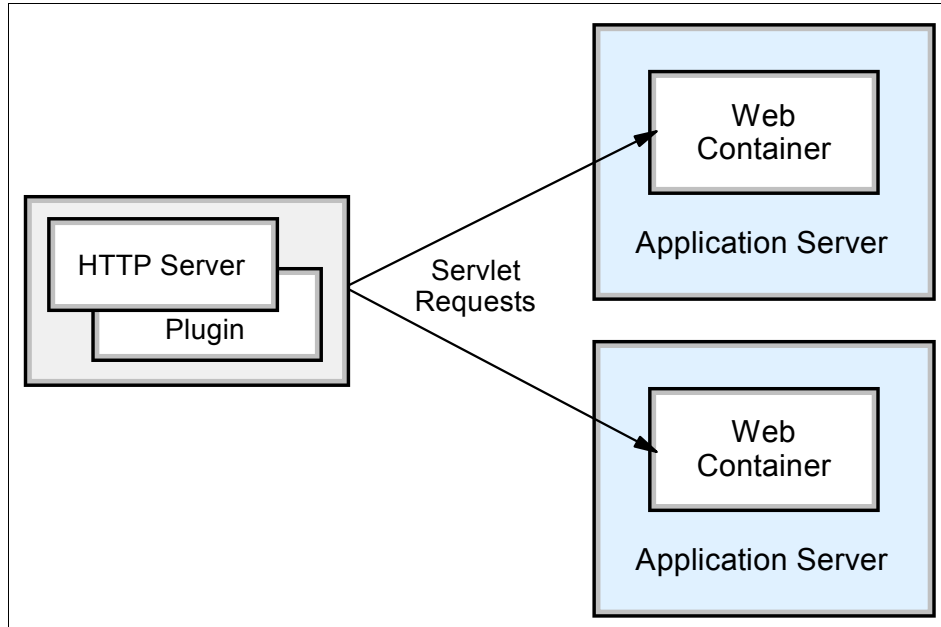


Figure 1-4 Plug-in (Web container) workload management

Clustering application servers that host Web containers automatically enables plug-in workload management for the application servers and the servlets they host. In the simplest case, the cluster is configured on a single machine, where the Web server process also runs.

Routing of servlet requests occurs between the Web server plug-in and the clustered application servers using HTTP or HTTPS. This routing is based purely on the weights associated with the cluster members. If all cluster members have identical weights, the plug-in will send equal requests to all members of the cluster assuming no strong affinity configurations. If the weights are scaled in the range from zero to 20, the plug-in will route requests to those cluster members with the higher weight value more often. A rule of thumb formula for determining routing preference would be:

$$\% \text{ routed to Server1} = \text{weight1} / (\text{weight1} + \text{weight2} + \dots + \text{weightn})$$

where there are  $n$  cluster members in the cluster. The Web server plug-in will temporarily route around unavailable cluster members.

See Chapter 5, “Plug-in workload management and failover” on page 133 for details.

### 1.3.3 Workload management using WebSphere clustering

This section describes how workload management is implemented in IBM WebSphere Application Server Network Deployment V5.1 by using application clusters and cluster members. How to create clusters and cluster members is detailed in Chapter 7, “Implementing the sample topology” on page 255.

A *cluster* is a set of application servers that are managed together and participate in workload management. Application servers participating in a cluster can be on the same node or on different nodes. A Network Deployment cell can contain no clusters, or have many clusters depending on the need of the administration of the cell.

The cluster is a logical representation of the application servers. It is not necessarily associated with any node, and does not correspond to any real server process running on any node. A cluster contains only application servers, and the weighted workload capacity associated with those servers.

When creating a cluster, it is possible to select an existing application server as the template for the cluster without adding that application server into the new cluster. This allows for continued development in the existing application, for example while using the definition for the cluster.

Cluster members can be added to a cluster in various ways - during cluster creation and afterwards. During cluster creation, one existing application server can be added to the cluster and/or one or more new application servers can be created and added to the cluster. There is also the possibility to add additional members to an existing cluster.

Cluster members are required to have identical application components, but can be sized differently in terms of weight, heap size, and other environmental factors. This allows large enterprise machines to belong to a cluster that also contains smaller machines such as Intel® based Windows® servers.

**Note:** The premise of the server group and clone from WebSphere V4.0 no longer applies to WebSphere Application Server V5.0 and higher. The cluster is a grouping of separately configured application servers that are serving the same application. The application servers themselves have no requirements for similarity outside of those components required for the application to run.

Starting or stopping the cluster will automatically start or stop all the cluster members, and changes to the application will be propagated to all application servers in the cluster.



Figure 1-5 shows an example of a possible configuration that includes server clusters. Server cluster 1 has two cluster members on node A and three cluster members on node B. Server cluster 2, which is completely independent of server cluster 1, has two cluster members on node B only. Finally, node A also contains a free-standing application server that is not a member of any cluster.

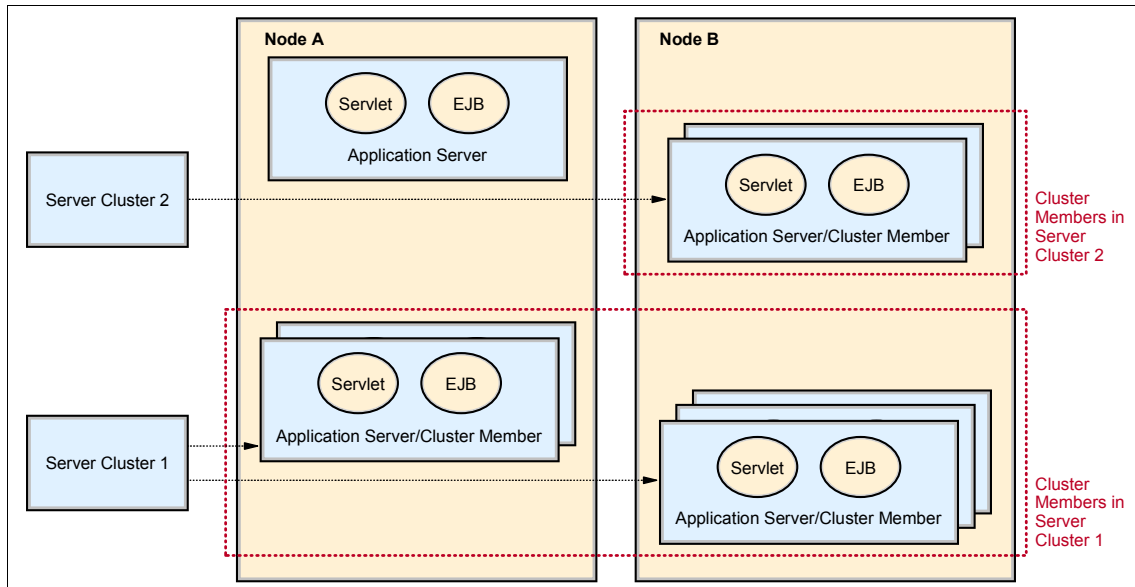


Figure 1-5 Server clusters and cluster members

Clusters and cluster members provide necessary support for workload management, failover, and scalability.

## Distributing workloads

The ability to route a request to any server in a group of clustered application servers allows the servers to share work and improving throughput of client requests. Requests can be evenly distributed to servers to prevent workload imbalances in which one or more servers have idle or low activity while others are overburdened. This load balancing activity is a benefit of workload management. Using weighted definitions of cluster members allows nodes to have different hardware resources and still participate in a cluster. The weight specifies that the application server with a higher weight will be more likely to serve the request faster, and workload management will consequently send more requests to that node.

## Failover

With several cluster members available to handle requests, it is more likely that failures will not negatively affect throughput and reliability. With cluster members distributed to various nodes, an entire machine can fail without any application downtime. Requests can be routed to other nodes if one node fails. Clustering also allows for maintenance of nodes without stopping application functionality.

## Scalability

Clustering is an effective way to perform vertical and horizontal scaling of application servers.

- In *vertical scaling*, shown in Figure 1-6, multiple cluster members for an application server are defined on the same physical machine, or node, which may allow the machine's processing power to be more efficiently allocated.

Even if a single JVM can fully utilize the processing power of the machine, you may still want to have more than one cluster member on the machine for other reasons, such as using vertical clustering for software failover. If a JVM reaches a table/memory limit (or if there is some similar problem), you can have another process to go to for failover.

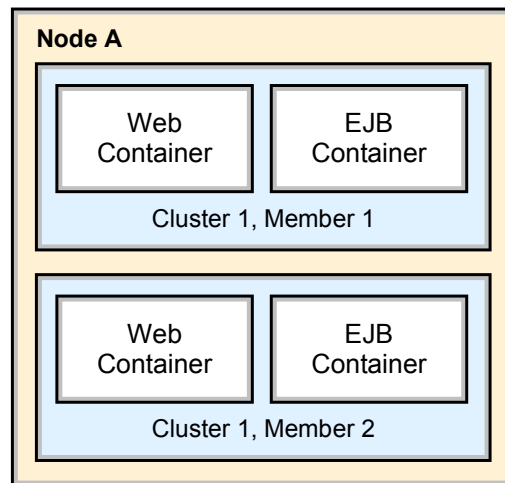


Figure 1-6 Vertical scaling

We recommend that you avoid using “rules of thumb” when determining the number of cluster members needed for a given machine. The only way to determine what is correct for your environment and application(s) is to tune a single instance of an application server for throughput and performance, then

add it to a cluster, and incrementally add additional cluster members. Test performance and throughput as each member is added to the cluster. Always monitor memory usage when you are configuring a vertical scaling topology and do not exceed the available physical memory on a machine.

In general, 85% (or more) utilization of the CPU on a large server shows that there is little, if any, performance benefit to be realized from adding additional cluster members.

- In *horizontal scaling*, shown in Figure 1-7, cluster members are created on multiple physical machines. This allows a single WebSphere application to run on several machines while still presenting a single system image, making the most effective use of the resources of a distributed computing environment. Horizontal scaling is especially effective in environments that contain many smaller, less powerful machines. Client requests that overwhelm a single machine can be distributed over several machines in the system. Failover is another benefit of horizontal scaling. If a machine becomes unavailable, its workload can be routed to other machines containing cluster members.

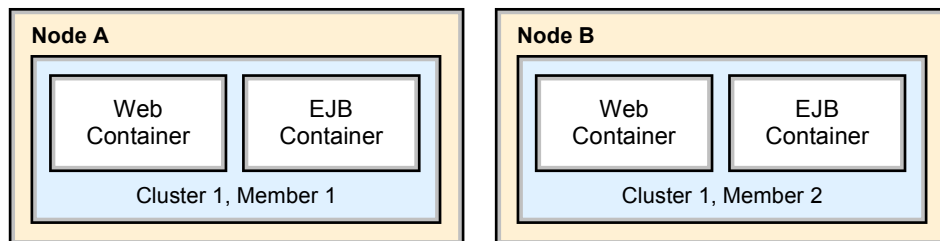


Figure 1-7 Horizontal scaling

Horizontal scaling can handle application server process failure and hardware failure without significant interruption to client service.

**Note:** WebSphere Application Server V5.0 and higher supports horizontal clustering across different platforms and operating systems. Horizontal cloning on different platforms was not supported in WebSphere Application Server V4.

- WebSphere applications can combine horizontal and vertical scaling to reap the benefits of both scaling techniques, as shown in Figure 1-8 on page 20.

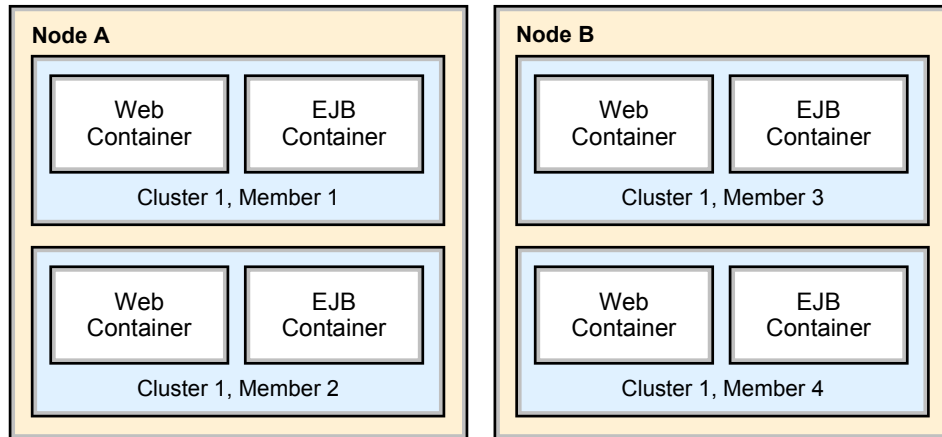


Figure 1-8 Vertical and horizontal scaling

### Secure application cluster members

The workload management service has its own built-in security, which works with the WebSphere Application Server security service to protect cluster member resources. If security is needed for your production environment, enable security before you create a cluster for the application server. This enables security for all of the members in that cluster.

The EJB method permissions, Web resource security constraints, and security roles defined in an enterprise application are used to protect EJBs and servlets in the application server cluster. Refer to *IBM WebSphere V5.0 Security*, SG24-6573 for more information.

## 1.3.4 Enterprise Java Services workload management

In this section, we discuss Enterprise Java Services workload management (EJS WLM), in which the Web container and EJB container are on different application servers. The application server hosting the EJB container is clustered.

Configuring the Web container in a separate application server from the Enterprise JavaBean container (an EJB container handles requests for both session and entity beans) enables distribution of EJB requests between the EJB container clusters, as seen in Figure 1-9 on page 21.

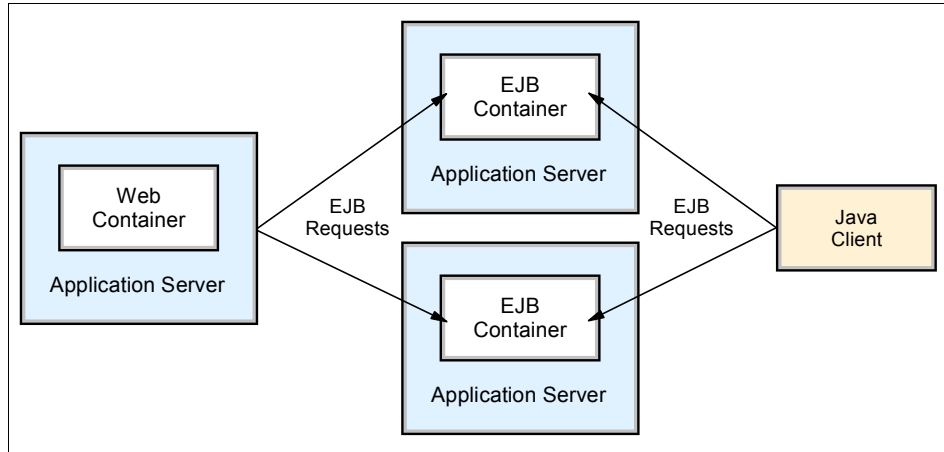


Figure 1-9 EJB workload management

In this configuration, EJB client requests are routed to available EJB containers based on the workload management EJB selection policy (Server-weighted round robin routing or Prefer local).

**Important:** Although it is possible to split the Web container and EJB container, it is not recommended because of the negative performance impact.

The EJB clients can be servlets operating within a Web container, stand-alone Java programs using RMI/IIOP, or other EJBs.

EJS workload management is covered in Chapter 6, “EJB workload management” on page 215.

## 1.4 Managing session state among servers

All the load distribution techniques discussed in this book rely, on one level or another, on using multiple copies of an application server and arranging for multiple consecutive requests from various clients to be serviced by different servers.

If each client request is completely independent of every other client request, then it does not matter if two requests are ever processed on the same server or not. However, in practice, there are many situations where all requests from an individual client are not totally independent. In many usage scenarios, a client makes one request, waits for the result, then makes one or more subsequent requests that depend upon the results received from the earlier requests.

Such a sequence of operations on behalf of one client falls into one of two categories:

- ▶ **Stateless:** the server that processes each request does so based solely on information provided with that request itself, and not based on information that it “remembers” from earlier requests. In other words, the server does not need to maintain state information between requests.
- ▶ **Stateful:** the server that processes a request *does* need to access and maintain state information generated during the processing of an earlier request.

Again, in the case of stateless interactions, it does not matter if different requests are being processed by different servers. But in the case of stateful interactions, we must ensure that whichever server is processing a request has access to the state information necessary to service that request. This can be ensured either by arranging for the same server to process all the client requests associated with the same state information, or by arranging for that state information to be shared and equally accessible by all servers that may require it. In that last case, it is often advantageous to arrange for most accesses to the shared state to be performed from the same server, so as to minimize the communications overhead associated with accessing the shared state from multiple servers.

This consideration for the management of stateful interactions will be a factor in many discussions of various load-distribution techniques throughout this book. It also requires the discussion of a specific facility, the session management facility, and of server affinity.

### 1.4.1 HTTP sessions and the session management facility

In the case of an HTTP client interacting with a servlet, the state information associated with a series of client requests is represented as an HTTP session, and identified by a session ID. The session manager module that is part of each Web container is responsible for managing HTTP sessions, providing storage for session data, allocating session IDs, and tracking the session ID associated with each client request, through the use of cookies, URL rewriting, or SSL ID techniques.

The session manager provides for the storage of session-related information either in-memory within the application server, in which case it cannot be shared with other application servers; in a back-end database, shared by all application servers; or by using memory-to-memory replication.

The second option, sometimes referred to as *persistent sessions* or *session clustering*, is one method that uses HTTP sessions with the load-distribution configuration. With this option, whenever an application server receives a request associated with a session ID, which is not in memory, it can obtain it by accessing the back-end database, and can then serve the request. When this option is not enabled, and another clustering mechanism is not used, if any load-distribution mechanism happens to route an HTTP request to an application server other than the one where the session was originally created, that server would be unable to access the session, and would thus not produce correct results in response to that request. One drawback to the database solution is that it provides a single point of failure (SPOF) that must be architected for with other products such as HACMP solutions and database replication. Another drawback is the performance hit, caused by database disk I/O operations and network communications.

The last option, memory-to-memory replication, provides a new mechanism for session clustering within IBM WebSphere Application Server Network Deployment V5. It uses the built-in Data Replication Service (DRS) of WebSphere V5 to replicate session information stored in memory to other members of the cluster. Using this functionality removes the single point of failure that is present in persistent sessions through a database solution that has not been made highly available using clustering software. The sharing of session state is handled by creating an internal replication domain, adding replication entries, and then configuring the Web container to use the domain entries and the replication for session state information.

**Important:** The latest performance measurements showed that the overhead associated with replicating the session to every other cluster member is more significant than contention introduced by using the database session repository (the more application servers in the cluster, the more overhead for memory-to-memory replication). However, these measurements were taken using the default memory-to-memory session replication configuration (that is peer-to-peer) which is not an optimal configuration.

Memory-to-memory replication performance can be greatly improved by using/configuring:

- ▶ Replication domains
- ▶ Domain partitioning
- ▶ DRS connection pooling
- ▶ Client/Server replication

For larger session sizes the overhead of session replication increases. Database replication has a lower overall throughput than memory-to-memory due to database I/O limitations (the database becomes the bottleneck). However, while database replication with large sessions performs slower, it is doing so with less CPU power than memory-to-memory replication. The unused processor power can be used by other tasks on the system.

Storing session states in a persistent database or using memory-to-memory replication provides a degree of fault tolerance to the system. If an application server crashes or stops, any session state that it may have been working on would normally still be available either in the back-end database or in the running application server's memory, so that other application servers can take over and continue processing subsequent client requests associated with that session.

**Important:** Neither mechanism provides a 100% guarantee that a session state will be preserved in case of a server crash. If a server happens to crash while it is literally in the middle of modifying the state of a session, some updates may be lost and subsequent processing using that session may be affected. Also, in the case of memory replication, if the node crashes during a replication, or in between the time interval that has been set for replicating session information, then some session information may be lost.

However, such a situation represents only a very small window of vulnerability, and a very small percentage of all occurrences that typically happen throughout the life of a system in production.

More information can be found in 5.5, "Session management" on page 166.



## 1.4.2 EJB sessions or transactions

In the case of an EJB client interacting with one or more EJBs, the management of state information associated with a series of client requests is governed by the EJB specification and implemented by the WebSphere EJS container, and depends on the types of EJBs that are the targets of these requests.

### Stateless session bean

By definition, when interacting with a stateless session bean, there is no client-visible state associated with the bean. Every client request directed to a stateless session bean is independent of any previous request that was directed to the same bean. The container will maintain a pool of instances of stateless session beans of each type, and will provide an arbitrary instance of the appropriate bean type whenever each client request is received. It does not matter if the same actual bean instance is used for consecutive requests, or even if two consecutive requests are serviced by bean instances in the same application server.

### Stateful session bean

In contrast, a stateful session bean is used precisely to capture state information that must be shared across multiple consecutive client requests that are part of one logical sequence of operations. The client must take special care to ensure that it is always accessing the same instance of the stateful session bean, by obtaining and keeping an EJB object reference to that bean. At the present time, WebSphere supports the distribution of stateful session bean homes among multiple application servers, but not the distribution of a specific instance. Each instance of a particular stateful session bean exists in only one application server, and can only be accessed by directing requests to that particular application server (of course, different instances of the same type of stateful session bean, used by different clients, may be spread among multiple servers). The various load-distribution techniques available in WebSphere make special provisions to support this characteristic of stateful session beans.

### Entity bean

Finally, we must consider the case of an entity bean. Most external clients access WebSphere services through session beans, but it is possible for an external client to access an entity bean directly. Furthermore, a session bean inside WebSphere is itself often acting as a client to one or more entity beans also inside WebSphere; if load-distribution features are used between that session bean and its target entity bean, then the same questions arise as with plain external clients.

Strictly speaking, the information contained in an entity bean is not usually associated with a “session” or with the handling of one client request or series of client requests. But it is common for one client to make a succession of requests targeted at the same entity bean instance. Unlike all the previous cases, it is possible for multiple independent clients to access the same entity bean instance more or less concurrently. Therefore, it is important that the state contained in that entity bean be kept consistent across the multiple client requests.

For entity beans, the notion of a session is more or less replaced by the notion of *transaction*. For the duration of one client transaction to which it participates, the entity bean is instantiated in one container (normally the container where the first operation within that transaction was initiated). All subsequent accesses to that same bean, within that same transaction, must be performed against that same instance in that same container.

In between transactions, the handling of the entity bean is specified by the EJB specification, in the form of a number of caching options:

- ▶ With option A caching, WebSphere Application Server assumes that the entity bean is used within a single container. Clients of that bean must direct their requests to the bean instance within that container. The entity bean has exclusive access to the underlying database, which means that the bean cannot be clustered or participate in workload management if option A caching is used.
- ▶ With option B caching, the bean instance remains active (so it is not guaranteed to be made passive at the end of each transaction), but it is always reloaded from the database at the start of each transaction. A client can attempt to access the bean and start a new transaction on any container that has been configured to host that bean.
- ▶ With option C caching (which is the default), the entity bean is always reloaded from the database at the start of each transaction and made passive at the end of each transaction. A client can attempt to access the bean and start a new transaction on any container that has been configured to host that bean. This is effectively similar to the session clustering facility described for HTTP sessions: the shared state is maintained in a shared database, and can be accessed from any server when required.

## Message-driven beans

The message-driven bean (MDB) was also introduced with WebSphere V5. MDB is a requirement of a J2EE 1.3 compliant application server. In WebSphere V4.0, the Enterprise Edition offered a similar functionality called message beans that leveraged stateless session EJBs and a message listener service. That container, however, did not implement the EJB 2.0 specification.

The MDB is a stateless component that is invoked by a J2EE container when a JMS message arrives at a particular JMS destination (either a queue or topic). Loosely, the MDB is triggered by the arrival of a message.

Messages are normally anonymous. If some degree of security is desired, the listener will assume the credentials of the application server process during the invocation of the MDB.

MDBs handle messages from a JMS provider within the scope of a transaction. If transaction handling is specified for a JMS destination, the listener will start a global transaction before reading incoming messages from that destination. Java Transaction API (JTA) transaction control for commit or rollback is invoked when the MDB processing has finished.

### 1.4.3 Server affinity

The discussion above implies that any load-distribution facility, when it chooses a server to direct a request, is not entirely free to select *any* available server:

- ▶ In the case of stateful session beans or entity beans within the context of a transaction, there is only one valid server. WebSphere WLM will always direct a client's access of a stateful session bean to the single server instance containing the bean (there is no possibility of choosing the wrong server here). If the request is directed to the wrong server (for example because of a configuration error), it will either fail, or that server itself will be forced to forward the request to the correct server, at great performance cost.
- ▶ In the case of clustered HTTP sessions or entity beans between transactions, the underlying shared database ensures that any server can correctly process each request. However, accesses to this underlying database may be expensive, and it may be possible to improve performance by caching the database data at the server level. In such a case, if multiple consecutive requests are directed to the same server, they may find the required data still in the cache, and thereby reduce the overhead of access to the underlying database.

The characteristics of each load-distribution facility, which take these constraints into account, are generally referred to as *server affinity*: In effect, the load distribution facility recognizes that multiple servers may be acceptable targets for a given request, but it also recognizes that each request may have a particular affinity for being directed to a particular server where it may be handled better or faster.

We will encounter this notion of server affinity throughout the discussion of the various load-distribution facilities in Chapter 5, “Plug-in workload management and failover” on page 133 and Chapter 6, “EJB workload management” on page 215. In particular, we will encounter the notion of *session affinity*, where the load-distribution facility recognizes the existence of a session and attempts to direct all requests within that session to the same server, and we will also discuss the notion of *transaction affinity*, in which the load-distribution facility recognizes the existence of a transaction, and behaves similarly.

Finally, we will also see that a particular server affinity mechanism can be weak or strong. In a weak affinity mechanism, the system attempts to enforce the desired affinity for the majority of requests most of the time, but may not always be able to provide a total guarantee that this affinity will be respected. In a strong affinity mechanism, the system guarantees that affinity will always be strictly respected, and generates an error when it cannot.

## 1.5 Performance improvements over previous versions

WebSphere Application Server V5.1 contains a significant set of improvements over the initial release of WebSphere Version 5.0 and the incremental releases that followed.

Performance improvements in V5.1 result from the following:

- ▶ Java version 1.4.1 - Provides significant performance improvement due to an improved JIT, new garbage collector features that allow more parallel processing, and other Java Virtual Machine (JVM) enhancements.
- ▶ EJB container and Transaction manager – Features a redesign of critical paths and improved persistence options for improved EJB performance due to reduced path length and more efficient algorithms.
- ▶ Web services – Provides performance improvements from enhancements, more efficient logic, and improved algorithms in:
  - Object Request Broker (ORB) class loading improvements
  - Performance advantages found in JDK 1.4.1
  - Web services engine performance improvements
  - Web container performance improvements
- ▶ Other - V5.1 contains miscellaneous performance improvements to the Servlet Engine and other WebSphere components.

Overall, performance gains for key design patterns were:

- ▶ EJB - average of 26% increase across all platforms
- ▶ Web Services - average of 30 to 50% for JavaBeans to EJB

- 50% faster, driving Web services to EJB primitives for large payloads
- 30% faster, driving Web services to JavaBeans primitives for large and small payloads

Existing customer applications will show improved performance, particularly applications that use:

- ▶ EJBs
- ▶ Messaging
- ▶ Web Services

## 1.6 The structure of this redbook

This redbook is divided into six parts containing the following chapters and appendixes:

### **Part 1: Getting started**

Chapter 2, “Design for scalability” on page 33 introduces scalability and scaling techniques, and helps you to evaluate the components of your e-business infrastructure for their scalability.

Chapter 3, “Introduction to topologies” on page 53 discusses topologies supported by WebSphere Application Server V5.x and the variety of factors that come into play when considering the appropriate topology choice.

### **Part 2: Distributing the workload**

Chapter 4, “Web server load balancing” on page 93 looks at Web server load balancing with the WebSphere Edge Components of IBM WebSphere Application Server Network Deployment V5.x, describing some basic configurations and its use with the IBM WebSphere Application Server Network Deployment V5.1. A short introduction to the Caching Proxy is also included in this chapter.

Chapter 5, “Plug-in workload management and failover” on page 133 covers Web container workload management. It discusses components, configuration settings, and resulting behaviors. It also covers session management in a workload-managed environment.

Chapter 6, “EJB workload management” on page 215 examines how EJB requests can be distributed across multiple EJB containers. Backup Cluster support (a function of IBM WebSphere Application Server Enterprise V5.1) is also explained in this chapter.

### **Part 3: Implementing the solution**

Chapter 7, “Implementing the sample topology” on page 255 provides step-by-step instructions for implementing a sample multi-machine environment. This environment is used to illustrate most of IBM WebSphere Application Server Network Deployment V5.1’s workload management and scalability features.

### **Part 4: High-availability solutions**

Chapter 8, “High availability concepts” on page 315 looks at the integration of platform clustering solutions with WebSphere to meet the high-availability needs of critical applications.

Chapter 9, “WebSphere Application Server failover and recovery” on page 345 looks at HTTP plug-in workload management and failover, HTTP session failover and behaviors, as well as considerations regarding the EJB container.

Chapter 10, “Deployment Manager and Node Agent high availability” on page 389 describes the options you have to make the administrative servers within WebSphere V5.0 highly available.

Chapter 11, “WebSphere Embedded JMS server and WebSphere MQ high availability” on page 417 describes techniques to cluster WebSphere MQ and Embedded JMS.

Chapter 12, “WebSphere data management high availability” on page 435 looks at various solutions to make the database highly available, such as WebSphere with HACMP for DB2 and Oracle, MC/ServiceGuard, Sun Cluster, VERITAS Cluster, DB2 Parallel Server, and many more.

Chapter 13, “High availability of LDAP, NFS, and firewall” on page 503 discusses configuration options for your load balancer, LDAP server, NFS, and firewall.

### **Part 5: Performance monitoring, tuning and coding practices**

Chapter 14, “Dynamic caching” on page 527 describes the use of caching technologies in three-tier and four-tier application environments involving browser-based clients and applications using WebSphere Application Server.

Chapter 15, “Understanding and optimizing use of JMS components” on page 591 describes how the various components for JMS interact, takes you through manually configuring the JMS components for the Trade3 application, and demonstrates running this on both an embedded JMS architecture and also with WebSphere MQ and WebSphere Business Integration Event broker.

Chapter 16, “Server-side performance and analysis tools” on page 685 explains the tools that can be run on the application server side to analyze and monitor the performance of the environment. The topics covered include the Performance

Monitoring Infrastructure, Tivoli® Performance Viewer, the Performance advisors, and the Technology Preview Tools that come with WebSphere Application Server V5.1.

Chapter 17, “Development-side performance and analysis tools” on page 759 offers an introduction into the tools that come with IBM WebSphere Studio Application Developer V5.1.1, such as the profiling tools or the downloadable Page Detailer.

Chapter 18, “Application development: Best practices for performance and scalability” on page 803 gives suggestions for developing WebSphere Application Server based applications.

Chapter 19, “Performance tuning” on page 821 discusses tuning an existing environment for performance and provides a guide for testing application servers and components of the total serving environment that should be examined and potentially tuned to increase the performance of the application

## **Part 6: Appendixes**

Appendix A, “Backup/recovery of Network Deployment configuration” on page 915 describes procedures to backup and restore the configuration data of a Network Deployment configuration, for example in the case of a disk failure.

Appendix B, “Sample URL rewrite servlet” on page 929 explains how to set up an example to test session management using URL rewrites.

Appendix C, “Additional material” on page 935 gives directions for downloading the Web material associated with this redbook.







## Design for scalability

Understanding the scalability of the components of your e-business infrastructure and applying appropriate scaling techniques can greatly improve availability and performance. Scaling techniques are especially useful in multi-tier architectures, when you want to evaluate components associated with dispatchers or edge servers, Web presentation servers, Web application servers, and data and transaction servers.

The approach to scaling your infrastructure presented in this chapter is based on the article “Design for Scalability - An Update”, available at:

<http://www.ibm.com/developerworks/websphere/library/techarticles/hvws/scalability.html>

## 2.1 Scaling your infrastructure

You can use the following high-level steps to classify your Web site and identify scaling techniques that are applicable to your environment:

1. Understand the application environment
2. Categorize your workload
3. Determine the most impacted components
4. Select the scaling techniques to apply
5. Apply the techniques
6. Reevaluate

This systematic approach needs to be adapted to your situation. We look at each step in detail in the following sections.

## 2.2 Understanding the application environment

For existing environments, the first step is to identify all components and understand how they relate to each other. The most important task is to understand the requirements and flow of the existing application(s) and what can or cannot be altered. The application is key to the scalability of any infrastructure, so a detailed understanding is mandatory to scale effectively. At a minimum, application analysis must include a breakdown of transaction types and volumes as well as a graphic view of the components in each tier.

Figure 2-1 on page 35 can aid in determining where to focus scalability planning and tuning efforts. The figure shows the greatest latency for representative customers in three workload patterns, and which tier the effort should be concentrated on.

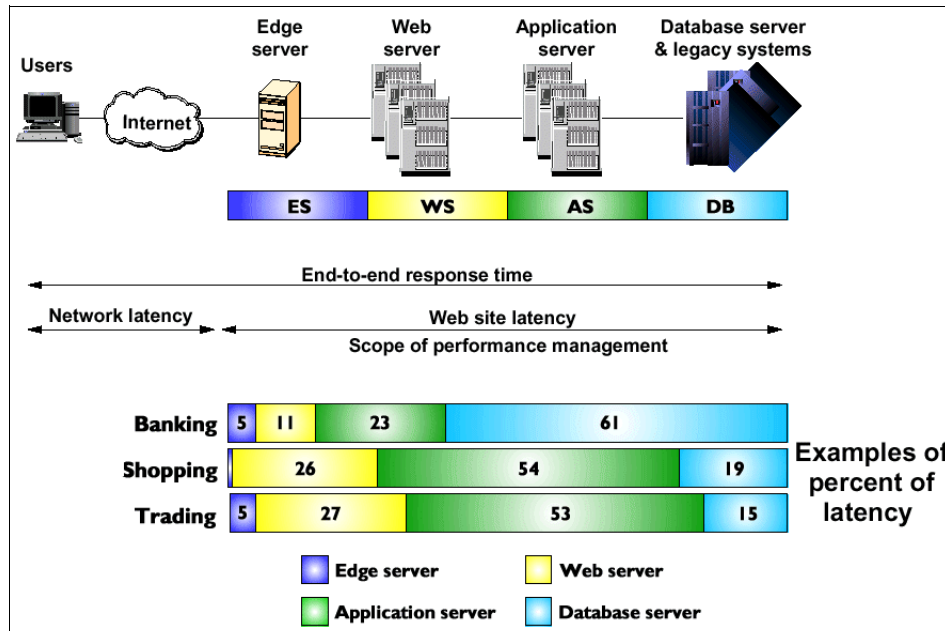


Figure 2-1 How latency varies based on workload pattern and tier

For example, for online banking, most of the latency typically occurs in the database server, whereas the application server typically experiences the greatest latency for online shopping and trading sites.

The way applications manage traffic between tiers significantly affects the distribution of latencies between the tiers, which suggests that careful analysis of application architectures is an important part of this step and could lead to reduced resource utilization and faster response times. Collect metrics for each tier, and make user-behavior predictions for each change implemented. WebSphere offers facilities for monitoring the performance of an application, as discussed in Chapter 16, “Server-side performance and analysis tools” on page 685 and Chapter 17, “Development-side performance and analysis tools” on page 759.

As requirements are analyzed for a new application, strive to build scaling techniques into the infrastructure. Implementation of new applications offer the opportunity to consider each component type, such as open interfaces and new devices, the potential to achieve unprecedented transaction rates, and the ability to employ rapid application development practices. Each technique affects

application design; similarly, application design impacts the effectiveness of the technique. To achieve proper scale, application design must consider potential scaling effects. An iterative, incremental approach will need to be followed in the absence of known workload patterns.

## 2.3 Categorizing your workload

Knowing the workload pattern (publish/subscribe or customer self-service, for example) determines where to focus scalability efforts, and which scaling techniques to apply. For example, a customer self-service site such as an online bank needs to focus on transaction performance and the scalability of databases that contain customer information used across sessions. These considerations would not typically be significant for a publish/subscribe site.

### 2.3.1 Workload patterns and Web site classifications

Web sites with similar workload patterns can be classified into site types. Consider five distinct workload patterns and corresponding Web site classifications:

- ▶ Publish/subscribe (user to data)
- ▶ Online shopping (user to online buying)
- ▶ Customer self-service (user to business)
- ▶ Online trading (user to business)
- ▶ Business to business

#### **Publish/subscribe (user to data)**

Sample publish/subscribe sites include search engines, media sites such as weather.com, and numerous newspapers and magazines, as well as event sites such as those for the Olympics and the Wimbledon championships.

Site content changes frequently, driving changes to page layouts. While search traffic is low in volume, the number of unique items sought is high, resulting in the largest number of page views of all site types. Security considerations are minor compared to other site types. Data volatility is low. This site type processes the fewest transactions and has little or no connection to legacy systems. See Table 2-1 on page 37.

Table 2-1 *Publish/subscribe site workload pattern*

Workload pattern	Publish/subscribe
Categories/examples	<ul style="list-style-type: none"> <li>▶ Search engines</li> <li>▶ Media</li> <li>▶ Events</li> </ul>
Content	<ul style="list-style-type: none"> <li>▶ Dynamic change of the layout of a page, based on changes in content, or need</li> <li>▶ Many page authors; page layout changes frequently</li> <li>▶ High volume, non-user-specific access</li> <li>▶ Fairly static information sources</li> </ul>
Security	Low
Percent secure pages	Low
Cross-session info	No
Searches	<ul style="list-style-type: none"> <li>▶ Structured by category</li> <li>▶ Totally dynamic</li> <li>▶ Low volume</li> </ul>
Unique Items	High
Data volatility	Low
Volume of transactions	Low
Legacy integration/ complexity	Low
Page views	High to very high

## Online shopping (user to online buying)

Sample sites include typical retail sites where users buy books, clothes, or even cars. Site content can be relatively static, such as a parts catalog, or dynamic, where items are frequently added and deleted, such as for promotions and special discounts that come and go. Search traffic is heavier than on a publish/subscribe site, although the number of unique items sought is not as large. Data volatility is low. Transaction traffic is moderate to high, and almost always grows.

When users buy, security requirements become significant and include privacy, non-repudiation, integrity, authentication, and regulations. Shopping sites have more connections to legacy systems, such as fulfillment systems, than publish/subscribe sites, but generally less than the other site types. This is detailed in Table 2-2.

*Table 2-2 Online shopping site workload pattern*

<b>Workload pattern</b>	<b>Online shopping</b>
Categories/examples	<ul style="list-style-type: none"> <li>▶ Exact inventory</li> <li>▶ Inexact inventory</li> </ul>
Content	<ul style="list-style-type: none"> <li>▶ Catalog either flat (parts catalog) or dynamic (items change frequently, near real time)</li> <li>▶ Few page authors and page layout changes less frequently</li> <li>▶ User-specific information: user profiles with data mining</li> </ul>
Security	<ul style="list-style-type: none"> <li>▶ Privacy</li> <li>▶ Non-repudiation</li> <li>▶ Integrity</li> <li>▶ Authentication</li> <li>▶ Regulations</li> </ul>
Percent secure pages	Medium
Cross-session info	High
Searches	<ul style="list-style-type: none"> <li>▶ Structured by category</li> <li>▶ Totally dynamic</li> <li>▶ High volume</li> </ul>
Unique items	Low to medium
Data volatility	Low
Volume of transactions	Moderate to high
Legacy integration/ complexity	Medium
Page views	Moderate to high

### **Customer self-service (user to business)**

Sample sites include those used for banking from home, tracking packages, and making travel arrangements. Home banking customers typically review their balances, transfer funds, and pay bills. Data comes largely from legacy applications and often comes from multiple sources, thereby exposing data consistency.

Security considerations are significant for home banking and purchasing travel services, less so for other uses. Search traffic is low volume; transaction traffic is moderate, but growing rapidly. See Table 2-3.

*Table 2-3 Customer self-service site workload pattern*

<b>Workload pattern</b>	<b>Customer self-service</b>
Categories/examples	<ul style="list-style-type: none"> <li>▶ Home banking</li> <li>▶ Package tracking</li> <li>▶ Travel arrangements</li> </ul>
Content	<ul style="list-style-type: none"> <li>▶ Data is in legacy applications</li> <li>▶ Multiple data sources, requirement for consistency</li> </ul>
Security	<ul style="list-style-type: none"> <li>▶ Privacy</li> <li>▶ Non-repudiation</li> <li>▶ Integrity</li> <li>▶ Authentication</li> <li>▶ Regulations</li> </ul>
Percent secure pages	Medium
Cross-session info	Yes
Searches	<ul style="list-style-type: none"> <li>▶ Structured by category</li> <li>▶ Low volume</li> </ul>
Unique items	Low
Data volatility	Low
Volume of transactions	Moderate and growing
Legacy integration/ complexity	High
Page views	Moderate to low

### **Online trading (user to business)**

Of all site types, trading sites have the most volatile content, the highest transaction volumes (with significant swing), the most complex transactions, and are extremely time sensitive. Auction sites are characterized by highly dynamic bidding against items with predictable life times.

Trading sites are tightly connected to the legacy systems. Nearly all transactions interact with the back-end servers. Security considerations are high, equivalent to online shopping, with an even larger number of secure pages. Search traffic is low volume. See Table 2-4.

*Table 2-4 Online trading site workload pattern*

<b>Workload pattern</b>	<b>Online trading</b>
Categories/examples	<ul style="list-style-type: none"> <li>▶ Online stock trading</li> <li>▶ Auctions</li> </ul>
Content	<ul style="list-style-type: none"> <li>▶ Extremely time sensitive</li> <li>▶ High volatility</li> <li>▶ Multiple suppliers, multiple consumers</li> <li>▶ Transactions are complex and interact with back end</li> </ul>
Security	<ul style="list-style-type: none"> <li>▶ Privacy</li> <li>▶ Non-repudiation</li> <li>▶ Integrity</li> <li>▶ Authentication</li> <li>▶ Regulations</li> </ul>
Percent secure pages	High
Cross-session info	Yes
Searches	<ul style="list-style-type: none"> <li>▶ Structured by category</li> <li>▶ Low volume</li> </ul>
Unique items	Low to medium
Data volatility	High
Volume of transactions	High to very high (very large swings in volume)
Legacy integration/ complexity	High
Page views	Moderate to high

## **Business to business**

These sites include dynamic programmatic links between arm's length businesses (where a trading partner agreement might be appropriate). One business is able to discover another business with which it may want to initiate transactions.



In supply chain management, for example, data comes largely from legacy applications and often from multiple sources, thereby exposing data consistency. Security requirements are equivalent to online shopping. Transaction volume is moderate, but growing; transactions are typically complex, connecting multiple suppliers and distributors. See Table 2-5.

*Table 2-5 Business-to-business site workload pattern*

<b>Workload pattern</b>	<b>Business to business</b>
Categories/examples	e-Procurement
Content	<ul style="list-style-type: none"> <li>▶ Data is in legacy applications</li> <li>▶ Multiple data sources, requirement for consistency</li> <li>▶ Transactions are complex</li> </ul>
Security	<ul style="list-style-type: none"> <li>▶ Privacy</li> <li>▶ Non-repudiation</li> <li>▶ Integrity</li> <li>▶ Authentication</li> <li>▶ Regulations</li> </ul>
Percent secure pages	Medium
Cross-session info	Yes
Searches	<ul style="list-style-type: none"> <li>▶ Structured by category</li> <li>▶ Low to moderate volume</li> </ul>
Unique items	Moderate
Data volatility	Moderate
Volume of transactions	Moderate to low
Legacy integration/ complexity	High
Page views	Moderate

### 2.3.2 Workload characteristics

Your site type will become clear as it is evaluated using Table 2-6 on page 42 for other characteristics related to transaction complexity, volume swings, data volatility, security, and so on.

If the application has further characteristics that could potentially affect its scalability, then add the extra characteristics to Table 2-6 on page 42.

**Note:** All site types are considered to have high volumes of dynamic transactions.

Table 2-6 Workload characteristics and Web site classifications

Workload characteristics	Publish/ sub- scribe	Online shopping	Customer self- service	Online trading	Business to business	Your workload
High volume, dynamic, transactional, fast growth	Yes	Yes	Yes	Yes	Yes	Yes
Volume of user-specific responses	Low	Low	Medium	High	Medium	
Amount of cross-session information	Low	High	High	High	High	
Volume of dynamic searches	Low	High	Low	Low	Medium	
Transaction complexity	Low	Medium	High	High	High	
Transaction volume swing	Low	Medium	Medium	High	High	
Data volatility	Low	Low	Low	High	Medium	
Number of unique items	High	Medium	Low	Medium	Medium	
Number of page views	High	Medium	Low	Medium	Medium	
Percent secure pages (privacy)	Low	Medium	Medium	High	High	
Use of security (authentication, integrity, non-repudiation)	Low	High	High	High	High	
Other characteristics						High

## 2.4 Determining the most affected components

This step involves mapping the most important site characteristics to each component. Once again, from a scalability viewpoint, the key components of the infrastructure are the dispatchers or edge servers, the Web application servers, security services, transaction and data servers, and the network. Table 2-7 on page 43 specifies the significance of each workload characteristic to each component. As seen in the table, the effect on each component is different for each workload characteristic.

**Note:** Whenever we use the terms dispatchers or edge servers throughout this chapter, we are referring to a technology that includes a variety of hardware and software solutions, one of which is the Load Balancer from the IBM WebSphere Application Server Network Deployment V5.1 Edge Components.

Table 2-7 Load impacts on components

Workload characteristics	Web present. server	Web app. server	Network	Security servers	Fire-walls	Existing business servers	Data-base server
High volume, dynamic, transactional, fast growth	High	High	High	High	High	High	High
High % user-specific responses	Low	High	Low	Low	Low	High	High
High % cross-session information	Med	High	Low	Low	Low	Low	Med
High volume of dynamic searches	High	High	Med	Low	Med	Med	High
High transaction complexity	Low	High	Low	Med	Low	High	High
High transaction volume swing	Med	High	Low	Low	Low	High	High
High data volatility	High	High	Low	Low	Low	Med	High
High # unique items	Low	Med	Low	Low	Low	High	High
High # page views	High	Low	High	Low	High	Low	Low
High % secure pages (privacy)	High	Low	Low	High	Low	Low	Low
High security	High	High	Low	High	Low	High	Low

## 2.5 Selecting the scaling techniques to apply

Best efforts in collecting the information needed are worthwhile in order to make the best possible scaling decisions. Only when the information gathering is as complete as it can be, then it is time to consider matching scaling techniques to components.

Manageability, security, and availability are critical factors in all design decisions. Techniques that provide scalability but compromise any of these critical factors cannot be used.

Here is a list of the eight scaling techniques:

- ▶ Use a faster machine
- ▶ Create a cluster of machines
- ▶ Use appliance servers
- ▶ Segment the workload
- ▶ Batch requests
- ▶ Aggregate user data
- ▶ Manage connections
- ▶ Cache

**Note:** Rather than buying hardware that can handle exponential growth that may or may not occur, consider specific approaches for these two types of servers:

- ▶ For application servers, the main technique for the growth path is to add more machines. It is therefore appropriate to start with the expectation of more than one application server with a dispatcher in front, such as the Load Balancer from the IBM WebSphere Application Server Network Deployment V5.1 Edge Components or WebSphere Edge Server. Adding more machines then becomes easier and far less disruptive.
- ▶ For data servers, get a server that is initially oversized; some customers run at just 30% capacity. This avoids the problem in some environments where the whole site can only use one data server. Another scaling option when more capacity is needed is to partition the database into multiple servers.

Many sites separate the application server from the database server. The front-end Web serving and commerce application functions are placed on less expensive commodity machines. The data server is placed on more robust and secure but more expensive systems.

## 2.5.1 Using a faster machine

This technique applies to the dispatchers or edge servers, the Web presentation server, the Web application server, the directory and security servers, the existing transaction and data servers, the network, and the Internet firewall.

The goal is to increase the ability to do more work in a unit of time by processing tasks more rapidly. Upgrading the hardware or software will result in a faster machine. However, one of the issues is that software capabilities can limit the hardware exploitation and vice versa. Another issue is that due to hardware or software changes, changes may be needed to existing system management policies.

**Tip:** Before replacing a machine, try to tune it. Unix operating systems are quite good tunable, Windows and Linux are less tunable.

For AIX tuning, refer to *Database Performance Tuning on AIX*, SG24-5511 or *IBM Certification Study Guide - AIX 5L Performance and System Tuning*, SG24-6184.

## 2.5.2 Creating a cluster of machines

This technique applies to the Web presentation server, the Web application server, and the directory and security servers. The primary goal here is to service more client requests. Parallelism in machine clusters typically leads to improvements in response time. Also, system availability is improved due to failover safety in replicas. The service running in a replica may have associated with it state information that must be preserved across client requests, and thus should be shared among machines.

State sharing is probably the most important issue with machine clusters and can complicate the deployment of this technique. WebSphere's workload balancing feature uses an efficient data-sharing technique to support clustering. Issues such as additional system management for hardware and software can also be challenging. IBM WebSphere Application Server Network Deployment V5.1 provides the WebSphere Edge Components that can be used for Web server load balancing/clustering and also allows you to create application server clusters. The clustering concept is discussed in various chapters throughout this book, for example Chapter 1, "Overview and key concepts" on page 3 and Chapter 4, "Web server load balancing" on page 93.

## 2.5.3 Using appliance servers

This technique applies to the edge servers, the Web presentation server, the directory and security servers, the network, and the Internet firewall. The goal is to improve the efficiency of a specific component by using a special-purpose machine to perform the required action. These machines tend to be dedicated machines that are very fast and optimized for a specific function. Examples are network appliances and routers with cache.

Some issues to consider regarding special machines are the sufficiency and stability of the functions and the potential benefits in relation to the added complexity and manageability challenges. It is worth noting, however, that devices of the newer generation are increasingly easy to deploy and manage; some are even self-managed.

## 2.5.4 Segmenting the workload

This technique applies to the Web presentation server, the Web application server, the data server, the intranet firewall, and the network. The goal is to split up the workload into manageable chunks, thereby obtaining a more consistent and predictable response time. The technique also makes it easier to manage which servers the workload is being placed on. Combining segmentation with replication often offers the added benefit of an easy mechanism to redistribute work and scale selectively, as business needs dictate.

An issue with this technique is that, in order to implement the segmentation, the different workloads serviced by the component need to be characterized. After segmenting the workload, additional infrastructure is required to balance physical workload among the segments. For more information about this technique, read Chapter 3, “Introduction to topologies” on page 53.

### **2.5.5 Batch requests**

This technique applies to the Web presentation server, the Web application server, the directory and security servers, the existing business applications, and the database. The goal is to reduce the number of requests sent between requesters and responders (such as between tiers or processes) by allowing the requester to define new requests that combine multiple requests.

The benefits of this technique arise from the reduced load on the responders by eliminating overhead associated with multiple requests. It also reduces the latency experienced by the requester due to the elimination of the overhead costs with multiple requests. These requests can be processed offline or at night time to reduce the load during peak hours.

Some of the issues are that there may be limits in achieving reuse of requests due to inherent differences in various requests types (for instance, Web front end differs from voice response front end). This can lead to increased costs of supporting different request types.

### **2.5.6 Aggregating user data**

This technique applies to the Web presentation server, the Web application server, and the network. The goal is to allow rapid access to large customer data controlled by existing system applications and support personalization based on customer specific data.

Accessing existing customer data spread across existing system applications may cause these applications to become overloaded, especially when the access is frequent. This can degrade response time. To alleviate this problem, the technique calls for aggregating customer data into a customer information service (CIS). A CIS that is kept current can provide rapid access to the customer data for a very large number of customers; thus, it can provide the required scalability.

An issue with a CIS is that it needs to scale very well to support large data as well as to field requests from a large number of application servers (requesters).

## 2.5.7 Managing connections

This technique applies to the Web presentation server, the Web application server, and the database. The goal is to minimize the number of connections needed for an end-to-end system, as well as to eliminate the overhead of setting up connections during normal operations. To reduce the overhead associated with establishing connections between each layer, a pool of pre-established connections is maintained and shared among multiple requests flowing between the layers.

For instance, WebSphere Application Server provides database connection managers to allow connection reuse. It is important to note that a session may use multiple connections to accomplish its tasks, or many sessions may share the same connection. This is called *connection pooling* in the WebSphere connection manager.

The key issue is with maintaining a session's identity when several sessions share a connection. Reusing existing database connections conserves resources and reduces latency for application requests, thereby helping to increase the number of concurrent requests that can be processed. Managing connections properly can improve scalability and response time. Administrators must monitor and manage resources proactively to optimize component allocation and use.

Refer to Chapter 19, “Performance tuning” on page 821 for more information.

## 2.5.8 Caching

Caching is a key technique to reduce hardware and administrative costs and to improve response time. Caching applies to the dispatcher or edge server, the Web presentation server, the Web application server, the network, the existing business applications, and the database. The goal is to improve performance and scalability by reducing the length of the path traversed by a request and the resulting response, and by reducing the consumption of resources by components.

Caching techniques can be applied to both static and dynamic Web pages. A powerful technique to improve performance of dynamic content is to asynchronously identify and generate Web pages that are affected by changes to the underlying data. Once these changed pages are generated, they must be effectively cached for subsequent data requests. There are several examples of intelligent caching technologies that can significantly enhance the scalability of e-business systems.

The key issue with caching dynamic Web pages is determining what pages should be cached and when a cached page has become obsolete.



**Note:** Caching is a technology that has several products available. One such product, the Caching Proxy, is part of the Edge Components available with IBM WebSphere Application Server Network Deployment V5.1. This chapter discusses in high-level terms some key concepts without subscribing to one product or another.

Information regarding caching techniques can be found in Chapter 4, “Web server load balancing” on page 93, Chapter 14, “Dynamic caching” on page 527, and Chapter 19, “Performance tuning” on page 821.

## 2.6 Applying the technique(s)

Apply the selected technique(s) in a test environment first, to evaluate not only the performance/scalability impact to a component, but also to determine how each change affects the surrounding components and the end-to-end infrastructure. A situation where improvements in one component result in an increased (and unnoticed) load on another component is undesirable.

Figure 2-2 on page 50 illustrates the typical relationship between the techniques and the key infrastructure components. By using this figure, the key techniques for each component can be identified.

In many cases, *all* techniques cannot be applied because one or more of the following is true:

- ▶ Investing in the techniques, even if it would prove helpful, is not affordable.
- ▶ There is no perceived need to scale as much as the techniques will allow.
- ▶ Cost/benefit analysis shows that the technique will not result in a reasonable payback.

Therefore, there must be a process for applying these techniques in different situations so that the best return is achieved. This mapping is a starting point and shows the components to focus on first, based on application workload.

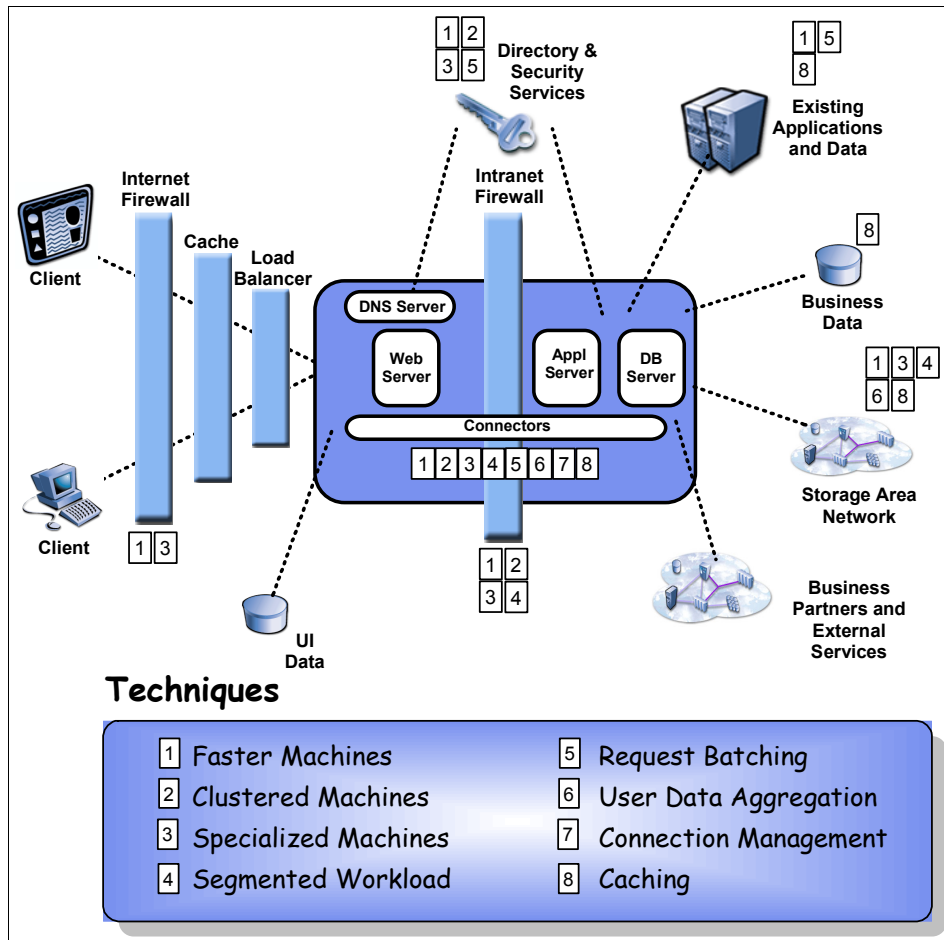


Figure 2-2 Scaling techniques applied to components

## 2.7 Re-evaluating

As with all performance-related work, tuning will be required. The goals are to eliminate bottlenecks, scale to a manageable status for those that cannot be eliminated, and work around those that cannot be scaled. Some examples of tuning actions and the benefits realized are:

- ▶ Increasing Web server threads raises the number of requests that could be processed in parallel.
- ▶ Adding indexes to the data server reduces I/O bottlenecks.

- ▶ Changing the defaults for several operating system variables allows threaded applications to use more heap space.
- ▶ Caching significantly reduces the number of requests to the data server.
- ▶ Increasing the number of edge/appliance servers improves load balancing.
- ▶ Upgrading the data server increases throughput.

Such results demonstrate the potential benefits of systematically applying scaling techniques and continuing to tune.

For an introduction into WebSphere tuning, refer to Chapter 19, “Performance tuning” on page 821.





# Introduction to topologies

This chapter discusses some basic topologies and a variety of factors that come into play when considering the appropriate choice, such as:

- ▶ Performance and throughput
- ▶ Scalability, availability, maintainability
- ▶ Security, session state

We describe some sample topologies that address these factors, presenting the fundamental advantages and disadvantages of each.

The following topics are discussed in this chapter:

- ▶ J2EE tiers model
- ▶ Topology selection criteria
- ▶ Strategies for scalability
- ▶ Single machine topology
- ▶ Separating the Web server
- ▶ Separating the database server
- ▶ Vertical scaling
- ▶ Horizontal scaling with clusters
- ▶ One WebSphere administrative cell versus many
- ▶ Multiple clusters on one node versus one cluster per node
- ▶ The sample topology
- ▶ Topologies and high availability
- ▶ Closing thoughts on topologies
- ▶ Topology selection summary

## 3.1 J2EE tiers model

This section covers the basics of a generic J2EE multi-tier application.

Java 2 Platform Enterprise Edition (J2EE) is a set of standards for developing and deploying enterprise Java applications, providing a multi-tier distributed application model, which can be divided basically into four main architectural tiers, as depicted in Figure 3-1:

- ▶ Client tier
- ▶ Presentation tier
- ▶ Business logic tier
- ▶ The Enterprise Information Systems (EIS) tier

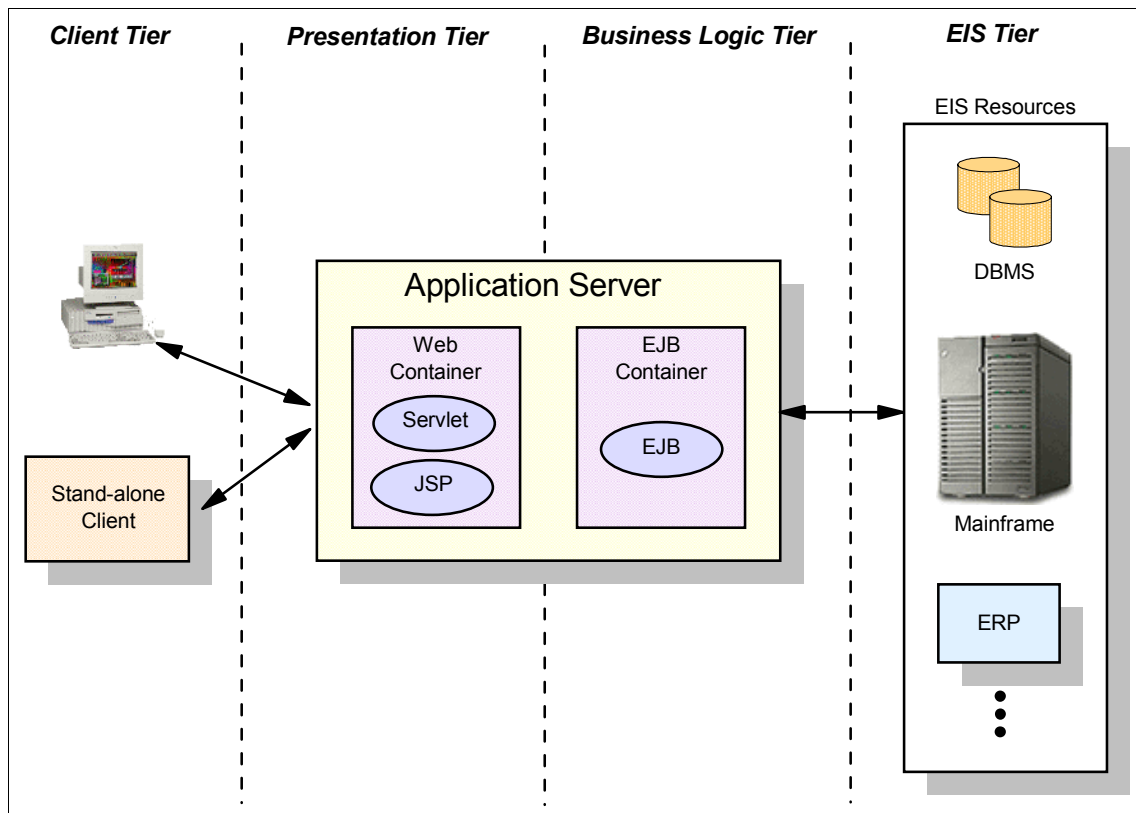


Figure 3-1 Multi-tier application model

The client tier encompasses various client types, such as browsers, applets or stand-alone application clients. These clients can reside both within and outside of the enterprise firewall. User actions are translated into server requests and the server responses are translated into a user-readable format.

The presentation tier embraces Web components, either JSP or servlets, which are deployed on Web containers. They access data and services from other tiers, handle user requests and screen flow, and can also control user interaction, presenting the returned information back to the client.

Business logic components access enterprise data and business rules, consisting of enterprise beans, deployed on EJB containers. There are three kinds of enterprise beans: session beans, entity beans, and message-driven beans.

The Enterprise Information Systems tier is commonly referred to as the back-end tier; examples include database manager systems, mainframe transaction processing and other legacy systems.

J2EE does not specify the structure and implementation of the runtime. It introduces the concept of container; the contract between applications and the container is specified via the J2EE APIs. WebSphere Application Server V5.1 delivers the infrastructure for deploying J2EE-compliant applications, providing the application servers on which we will run our Java programs. The application server implements the Web container and EJB container components.

On a WebSphere topology, the basic components that interact to execute our application are:

- ▶ The HTTP server and WebSphere Web server plug-in
- ▶ WebSphere Application Server (Web containers and EJB containers)
- ▶ Databases (application databases)

**Note:** The Web container in IBM WebSphere Application Server V5.1 has an embedded HTTP transport, which allows for direct connection to the application without the need for a separate Web server. While use of this transport as a Web server is very useful for testing or development purposes it should not be used in production environments. For performance and security reasons, it is recommended that you use a standalone Web server and HTTP plug-in for the Web server in a production environment.

The emphasis of our topologies scenarios will be the mapping of the J2EE application architecture tiers to a physical deployment on WebSphere Application Server as well as how to apply different techniques and component associations in order to provide scalability, load balancing and failover, based on multiple criteria.

## 3.2 Topology selection criteria

While a variety of factors come into play when considering the appropriate topology for a WebSphere deployment (see Chapter 1, “Overview and key concepts” on page 3), the primary factors to plan for typically include:

- ▶ Security
- ▶ Performance
- ▶ Throughput
- ▶ Scalability
- ▶ Availability
- ▶ Maintainability
- ▶ Session state

**Note:** For each of the common topologies, a decision needs to be made regarding the placement of the Deployment Manager and master cell repository. The Deployment Manager can be located either on a dedicated machine, or on the same machine as one of its nodes. The advantages and disadvantages of each approach are outlined in 3.2.3, “Coexistence of WebSphere Application Server and Network Deployment” of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195.

## 3.3 Strategies for scalability

On demand computing requires the ability to scale up or scale down an application, depending on the current requirements. Thus, scalability is important to improve efficiency and reduce cost.

We start by discussing scalability strategies using WebSphere Application Server that can help us in ensuring high availability, load balancing, and removing bottlenecks.

The basic infrastructure components that make up a WebSphere application are:

- ▶ HTTP server and HTTP plug-in
- ▶ Web container
- ▶ EJB container
- ▶ Databases

IBM WebSphere Application Server V5.0 and higher implements Web containers and EJB containers in each application server. The application servers each run in their own JVM (Java Virtual Machine).



If we have all components co-located on a single machine, they might:

- ▶ Compete for machine resources
- ▶ Influence each other's behavior
- ▶ Have unauthorized access to strategic resources

One strategy is to physically separate some components, preventing them from competing for resources (CPU, memory, I/O, network, and so on) or to restrict the access to a resource from another machine (for example, inserting a firewall in order to protect confidential information). This approach is represented in Figure 3-2.

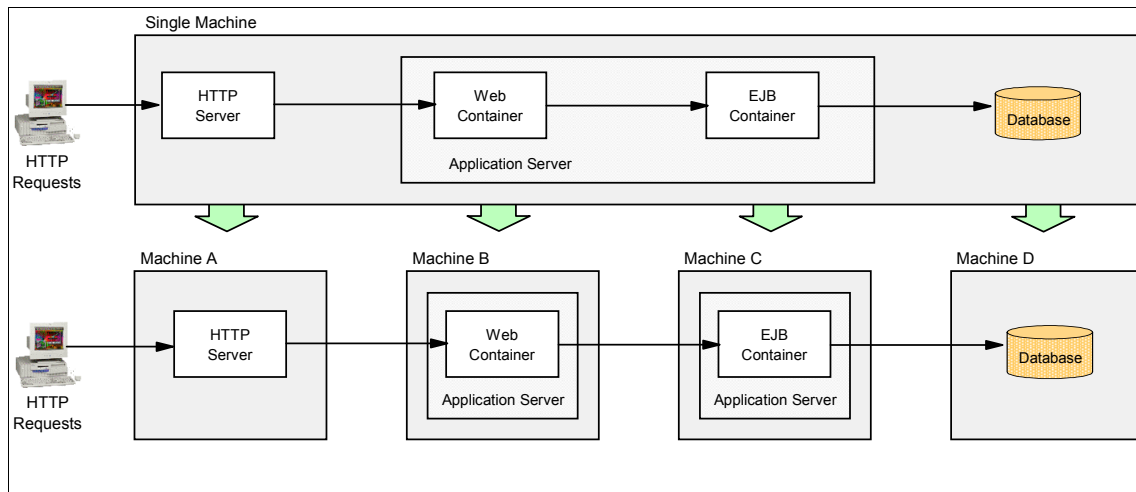


Figure 3-2 Separating components across multiple machines

**Attention:** Figure 3-2 shows the Web container and the EJB container separated on different machines. Although this is a possible configuration, it is not recommended because doing so results in out of process calls from the EJB clients in the Web container to the EJB container and will likely have a negative impact on performance. As a result, we will not cover this split JVM topology further in this chapter.

We can also exploit another strategy, distributing the load among the most appropriate resources, and using workload management techniques such as vertical and horizontal scaling, as described in 1.3.3, “Workload management using WebSphere clustering” on page 16. WebSphere Application Servers can benefit from vertical and horizontal scaling and the HTTP servers can be horizontally scaled on a clustered configuration. The use of these techniques is represented in Figure 3-3 on page 58.

Starting with 3.4, “Single machine topology” on page 60, we describe some basic topologies, beginning with a single machine topology and expanding it to a complex topology where different techniques are applied on a set of distributed machines, in order to provide a reliable and efficient processing environment.

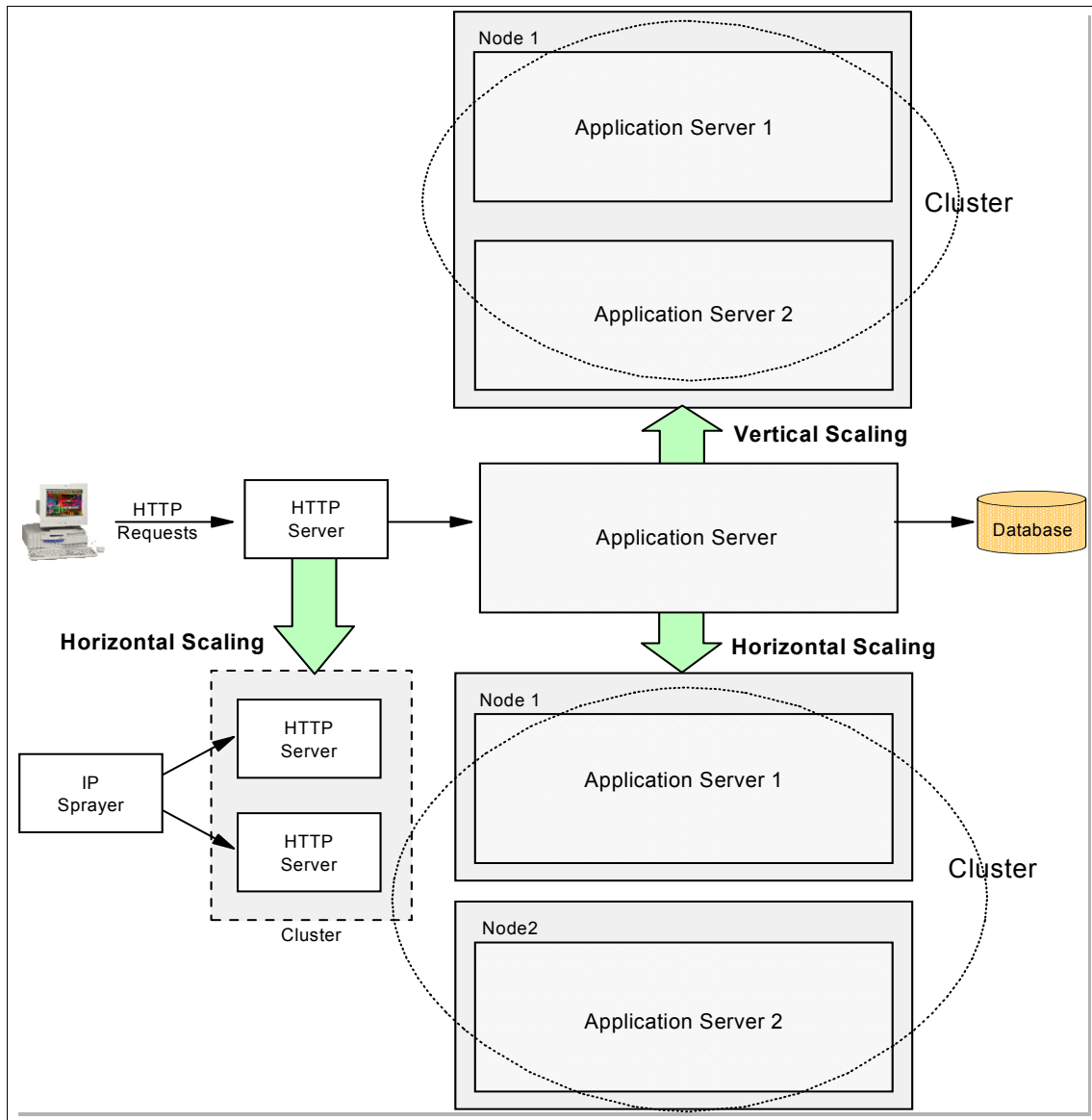


Figure 3-3 Vertical and horizontal scaling

## Session persistence considerations

If the application maintains state between HTTP requests and we are using vertical or horizontal scaling, then we must consider using an appropriate strategy for session management.

Each application server runs in its own JVM process. To allow a failover from one application server to another without logging out users, we need to share the session data between multiple processes. There are two different ways of doing this in IBM WebSphere Application Server Network Deployment V5.1:

- ▶ **Memory-to-memory session replication**

Provides replication of session data between the process memory of different application server JVMs. A Java Message Service (JMS) based publish/subscribe mechanism, called Data Replication Service (DRS), is used to provide assured session replication between the JVM processes. DRS is included with IBM WebSphere Application Server Network Deployment V5.1 and is automatically started when the JVM of a clustered (and properly configured) application server starts. Please note that DRS is not based on JMS, so you do not need to install the WebSphere JMS Provider.

- ▶ **Database persistence**

Session data is stored in a database shared by all application servers.

Memory-to-memory replication has the following advantages and disadvantages compared to database persistence:

- ▶ No separate database product is required. But since you normally will use a database product anyway for your application, this might not be an inhibitor.
- ▶ In the default peer to peer configuration, if there is a JVM memory constraint, using database session persistence rather than memory-to-memory replication is the better solution. However, there are also several options to optimize memory-to-memory replication, that allow for distributing the session objects to one or more application servers that are dedicated to serving as session object servers.

**Note:** See “HTTP sessions and the session management facility” on page 22 for additional information about the performance of database persistence versus memory-to-memory replication.

Refer to Chapter 14, “Configuring session management”, especially Section 14.9, “Persistent session management”, of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195 for detailed information.

## 3.4 Single machine topology

The starting scenario for our discussion about topologies is the configuration where all components reside on the same machine, as shown in Figure 3-4. The Web server routes requests, as appropriate, to the WebSphere Application Server on the same machine for processing.

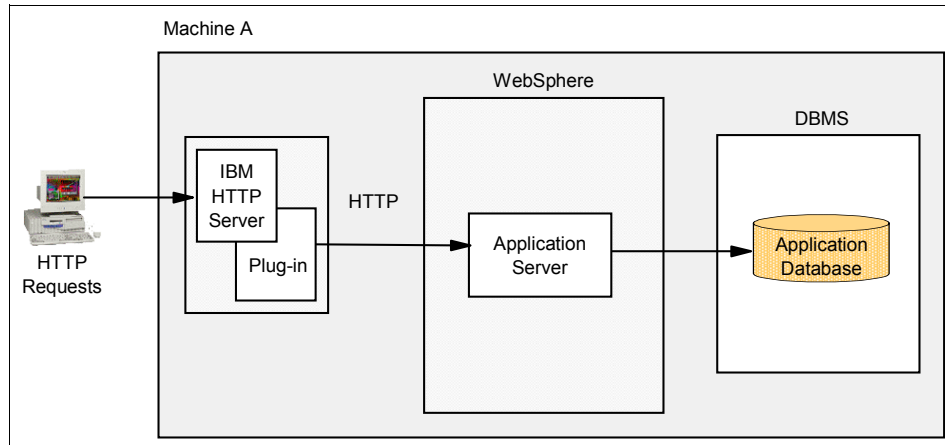


Figure 3-4 Single machine topology

Some reasons to use a single machine topology are:

- ▶ **Maintainability:** easy to install and maintain  
This configuration is most suitable as a startup configuration in order to evaluate and test the basic functionalities of WebSphere and related components. The installation is automated by tools supplied with the WebSphere distribution. This configuration is also the easiest to administer.
- ▶ **Performance, security and availability are not critical goals**  
This may be the case for development, testing, and some intranet environments. We are limited to the resources of a single machine, which are shared by all components.
- ▶ **Low cost**

Consider the following when you use a single machine topology:

- ▶ **Performance:** components' interdependence and competition for resources  
All components compete for the shared resources (CPU, memory, network, and so on). Since components influence each other, bottlenecks or ill-behaved components can be difficult to identify and remedy.

- ▶ Security: no isolation  
There is no explicit layer of isolation between the components.
- ▶ Availability: single point of failure  
This configuration is a single point of failure.

## 3.5 Separating the Web server

When compared to a configuration where the application server and the HTTP server are collocated on a single physical server, separation of the application server and the HTTP server can be utilized to provide varying degrees of improvement in:

- ▶ Performance
- ▶ Process isolation
- ▶ Security

This configuration is illustrated in Figure 3-5.

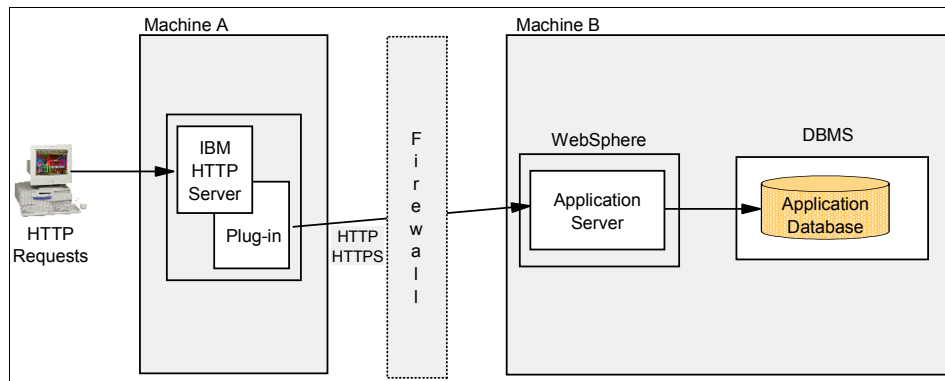


Figure 3-5 HTTP server separated from application server

The WebSphere V5.1 Web server plug-in allows the HTTP server to be physically separated from the application server. It uses an XML configuration file (plugin-cfg.xml) containing settings that describe how to handle and pass on requests, via a new connection, to the WebSphere Application Server(s).

**Note:** J2EE specifies that static Web content be packaged in the .war (Web archive) file, which means that it needs to be read and served by the WebSphere file serving servlet. This approach can be inefficient and is not recommended for a production environment.

In some cases you might want to “explode” the .war and install some of the static content on the Web server itself. In doing so, remember that you cannot secure this content with WebSphere security and you will have to adjust the plug-in configuration manually.

In WebSphere Application Server V5.1 serving files can be enabled/disabled during the creation of a Web module either through the File Server Enabled check box in the IBM extensions pane of the Application Server Toolkit (ASTK), or by changing the fileServingEnabled flag in the ibm-web-ext.xml file, located in the WEB-INF directory of the installed module.

Please note the ASTK is the WebSphere Application Server V5.1 replacement for the Application Assembly Tool (AAT) that shipped with prior versions of WebSphere. In WebSphere Application Server V5.1 the ASTK ships as a separate install image in addition to the application server.

In WebSphere V5.0.2 the ASTK is a technology preview available as an alternative to the AAT (Application Assembly Toolkit) as a separate download at:

[http://www.ibm.com/support/docview.wss?rs=180&context=SSEQTP&q=&uid=swg24005125&loc=en\\_US&cs=utf-8&lang=en-en](http://www.ibm.com/support/docview.wss?rs=180&context=SSEQTP&q=&uid=swg24005125&loc=en_US&cs=utf-8&lang=en-en)

Some reasons to separate the HTTP server from the application server are:

- ▶ Performance: size and configure servers appropriate to each task  
By installing components (Web server and application server) on separate machines, each machine can be sized and configured to optimize the performance of each component (for example, an application server might require faster and more powerful machines than the Web server system).
- ▶ Performance: remove resource contention  
By installing the Web server on a separate machine from the WebSphere Application Server, a high load of static requests will not affect the resources (CPU, memory, disk) available to WebSphere, and therefore will not affect its ability to service dynamic requests. The same applies when the Web server serves dynamic content using other technologies, such as CGI, ASP, and so on.

- Maintainability: component independence

Server components can be reconfigured, or even replaced, without affecting the installation of other components on separate machines.

- Security: separate Web server and WebSphere interfaces

In order to protect application servers from unauthorized outside access, the separation of the Web server from the application server is often used with firewalls to create a secure demilitarized zone (DMZ) surrounding the Web server. Isolating the HTTP server on a DMZ protects application logic and data by restricting the access from the public Web site to the servers and databases where this valuable information is stored. Desirable topologies should not have databases, nor servers that directly access databases, in the DMZ. WebSphere Application Server stores the configuration data as XML files. Furthermore, an application installed on WebSphere usually needs to access a database. Therefore, it is often not an acceptable solution to run WebSphere Application Server in the DMZ.

Consider the following when you separate the HTTP server from the application server:

- Maintainability: configuration complexity

The configuration file (plugin-cfg.xml) is generated on the WebSphere Application Server machine and must then be copied to the Web server machine.

- Performance: network access may limit performance

Depending upon the network capacity and remoteness of the Web server, the network response time for communication between WebSphere Application Server and the Web server may limit the application response time. To prevent this you should insure that you have adequate network bandwidth between the Web server and the application server. When colocated on the same server, network response is usually not an issue, but other resource constraints such as memory and CPU can limit performance.

- Security: encrypted transport

The HTTP plug-in also allows encryption of the link between the Web server and the application server, using HTTP over SSL (HTTPS) data encryption. This reduces the risk that attackers would be able to obtain secure information by “sniffing” packets sent between the Web server and application server. A performance penalty usually accompanies such encryption. The HTTP plug-in is suitable for environments that require all network communication to be encrypted, even in a DMZ.

It is recommended that this connection can be configured so that the HTTPS plug-in and Web container must mutually authenticate each other using public-key infrastructure (PKI). This prevents unauthorized access to the Web container.

- Security: access to protected resources

The plug-in is also compatible with the WebSphere Application Server V5.1 security model, allowing access to protected applications and their resources while enforcing authorization and authentication policies.

- Security: allows the use of NAT

Some firewall environments use Network Address Translation (NAT). NAT receives packets for one IP address and translates the headers of the packet so it can be sent to a second IP address. In this case, we should avoid using configurations involving complex protocols in which IP addresses are embedded in the body of the IP packet, such as Java Remote Method Invocation (RMI) or Internet Inter-ORB Protocol (IIOP). Most firewalls cannot translate packets for these protocols, as a result the IP addresses are not translated, making the packet useless. Using HTTP as the transport between the plug-in and the application server avoids these problems associated with NAT and complex protocols.

- Security: number of opened ports

Using HTTP requests to send data between the plug-in and the application server through the firewall requires opening only one port. Configurations that require switching to another protocol (such as IIOP), and opening firewall ports corresponding to the protocol, are less desirable. They are often more complex to set up, and the protocol switching overhead can impact performance.



**Notes:**

- ▶ The servlet redirector alternatives for separating the HTTP server from the application server, available in WebSphere V3.x, are no longer supported. The unsupported servlet redirector alternatives include:
  - Thick Servlet Redirector
  - Thick Servlet Redirector administrative server agent
  - Thin Servlet Redirector
- ▶ The OSE plug-in provided in previous releases of WebSphere Application Server is not supported either. All installations should now use the new HTTP plug-in.
- ▶ The reverse proxy approach for separating the HTTP server and the application server is generally not suggested for use with WebSphere V5.0 and higher. The HTTP plug-in behaves much like a reverse HTTP proxy, without its disadvantages.

Reverse proxy disadvantages include:

- Extra hardware and software required
- Not WebSphere workload management aware

However, the use of a reverse proxy has benefits if placed before the HTTP server. See 3.3.4, “Topology 3: reverse proxy” in the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195 for more information.

## 3.6 Separating the database server

WebSphere is usually configured to access the database server, for example DB2 Server Enterprise Edition or Oracle 8i, through TCP/IP. As long as the host name, port number and database ID settings are correct, WebSphere will be able to communicate with a database server located anywhere accessible through TCP/IP.

In the simple single machine configuration described in 3.4, “Single machine topology” on page 60, the WebSphere application database and WebSphere Application Server reside on the same server. However, installing the database on a different machine, creating a two-tier configuration as illustrated in Figure 3-6 on page 66, represents a good practice, with several advantages.

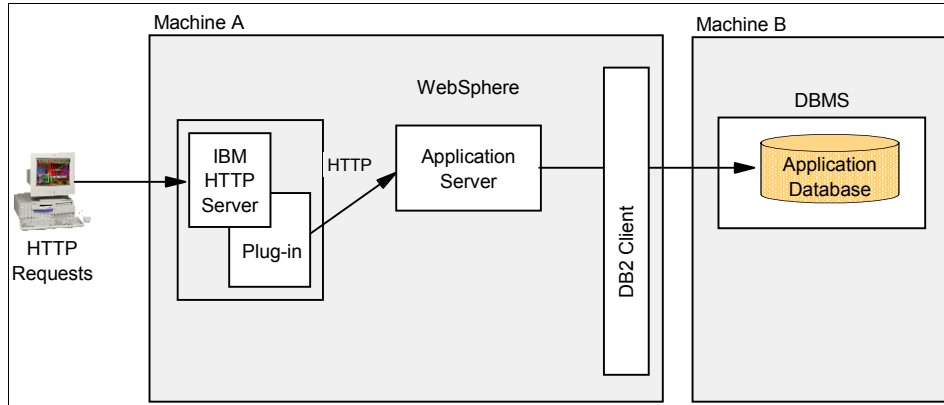


Figure 3-6 Separating the database server

All remote database clients use a communications product to support the protocol used to access a remote database server. The protocol stack must be installed and configured before a client can communicate with a remote DB2 UDB server. For WebSphere Application Server, the recommended protocol is TCP/IP. See the WebSphere Application Server installation documentation for your platform for instructions on how to configure a remote DB2 database.

Some reasons to separate the database server are:

- Performance: less competition for resources

If both the database and application server are placed on the same machine, then under high load, you have two resource intensive processes: the application server and the database server, competing for increasingly scarce resources (CPU and memory). So, in general, we can expect significantly better performance by separating the application server from the database server.

- Performance: differentiated tuning

By separating the servers, we can independently tune the machines that host the database server and the application server to achieve optimal performance. The database server is typically sized and tuned for database performance, which may differ from the optimal configuration for the application server.

On many UNIX® servers, installing the database involves modification of the operating system kernel. For example, the installation and execution of Oracle 8i on Solaris requires tuning of the kernel settings to meet Oracle requirements. This database-specific tuning can be detrimental to the performance of application servers located on the same machine.

- ▶ **Availability:** use of already established highly available database servers  
Many organizations have invested in high-availability solutions for their database servers, reducing the possibility of the server being a single point of failure in a system.
- ▶ **Maintainability:** independent installation/reconfiguration  
Components can be reconfigured, or even replaced, without affecting the installation of the other component.

Consider the following when using a remote database server:

- ▶ **Network access may limit performance**  
Insure that adequate network bandwidth exists between the application server and the database server otherwise the network response time for communication between WebSphere Application Server and the database server may limit the performance of WebSphere. When colocated on the same server, network response is usually not an issue, instead performance and scalability is limited by resource constraints (memory and CPU).
- ▶ **Architectural complexity**  
Hosting the database server on a separate machine introduces yet another box that must be administered, maintained, and backed up.
- ▶ **Maintainability: complexity of configuration**  
Remote database access requires more complex configuration, setting up clients, and so on.
- ▶ **Cost**  
The cost of a separate machine for database server may not be justified for the environment in which the WebSphere Application Server will be installed. However, this is typically not an issue since database servers are usually shared resources for many different applications and application clients, which may or may not be running on WebSphere.

## 3.7 Vertical scaling

Vertical scaling provides a straightforward mechanism for creating multiple instances of an application server, and hence multiple JVM processes.

In the simplest case, one can create many application servers on a single machine, and this single machine also runs the HTTP server process. This configuration is illustrated in “Vertical scaling with cluster” on page 68.

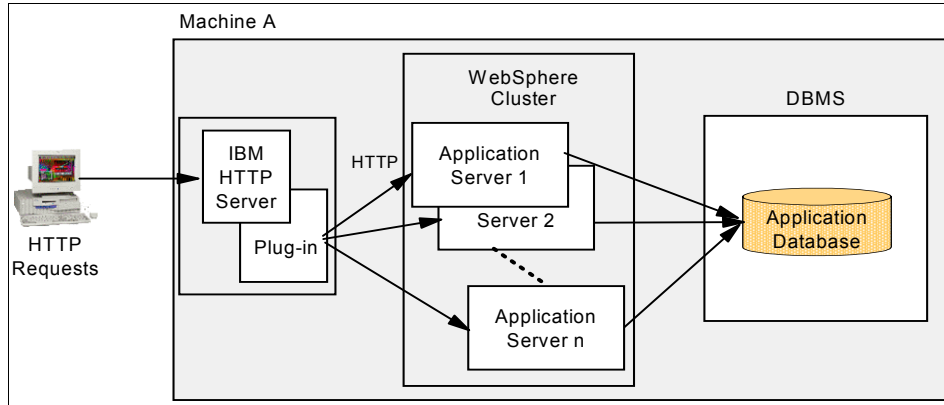


Figure 3-7 Vertical scaling with cluster

To set up a vertical scaling topology, refer to Chapter 7, “Implementing the sample topology” on page 255.

It is recommended that you plan vertical scaling configurations ahead of time. However, since they do not require any special installation steps, you can always implement them later on an as-needed basis.

Although there is a limit to the resources available on a single machine, and an availability risk when using a single machine, this configuration provides process isolation. It may also provide improved throughput over a configuration where only a single application server process is running.

This assumes, of course, that sufficient CPU and memory are available on the machine (and that the application can take advantage of such a configuration). Therefore, when implementing a vertically scaled WebSphere environment, make sure you do not exceed the available resources.

**Important:** The best way to ensure good performance in a vertical scaling configuration is to tune a single instance of an application server for throughput and performance, then incrementally add additional instances. Test performance and throughput as each application server instance is added. Once CPU utilization is 85% or greater, it's likely that adding additional instances will do little to improve performance and throughput as the operating system starts performing context switching between processes. Always monitor memory use when you are configuring a vertical scaling topology and do not exceed the available physical memory on a machine.

Some reasons to use vertical scaling are:

- Performance: better use of the CPU

An instance of an application server runs in a single Java Virtual Machine (JVM) process. However, the inherent concurrency limitations of an application in the JVM process may prevent it from fully utilizing the processing power of a machine. Creating additional JVM processes provides multiple thread pools, each corresponding to the JVM associated with each application server process. This can enable the application in the application server(s) to make the best possible use of the processing power and throughput of the host machine.

- Availability: failover support in a cluster

A vertical scaling topology also provides process isolation and failover support within an application server cluster. If one application server instance goes offline, the requests to be processed will be redirected to other instances on the machine.

- Throughput: WebSphere workload management

Vertical scaling topologies can make use of the WebSphere Application Server workload management facility. The HTTP server plug-in distributes requests to the Web containers and the ORB distributes requests to EJB containers.

- Maintainability: easy to administer member application servers

With the concept of cells and clusters in WebSphere Application Server V5, it is easy to administer multiple application servers from a single point.

- Maintainability: vertical scalability can easily be combined with other topologies

We can implement vertical scaling on more than one machine in the configuration (IBM WebSphere Application Server Network Deployment V5.1 must be installed on each machine). We can combine vertical scaling with the other topologies described in this chapter to boost performance and throughput.

Again, this assumes, of course, that sufficient CPU and memory are available on the machine.

- Cost: does not require additional machines

Consider the following when using vertical scaling:

- Availability: machine still a single point of failure

Single machine vertical scaling topologies have the drawback of introducing the host machine as a single point of failure in the system. However, this can be avoided by using vertical scaling on multiple machines.

- Performance: scalability limited to a single machine

Scalability is limited to the resources available on a single machine.

When you are deciding how many application servers to create on a machine, you need to take several factors into account:

- Applications that make use of more components require more memory, limiting the number of servers that can be run on a machine.
- Vertical scaling is best performed on machines with plenty of memory and processing power. However, eventually the overhead of running more application servers cancels out the benefits of adding them.

**Note:** Version 1.2 and above of the IBM Java 2 Software Development Kit (SDK) handles parallel JVM processes better than earlier versions.

## 3.8 Horizontal scaling with clusters

Horizontal scaling exists when the members of an application server cluster are located across multiple physical machines. This lets a single WebSphere application span several machines, yet still present a single logical image. This configuration is illustrated in Figure 3-8.

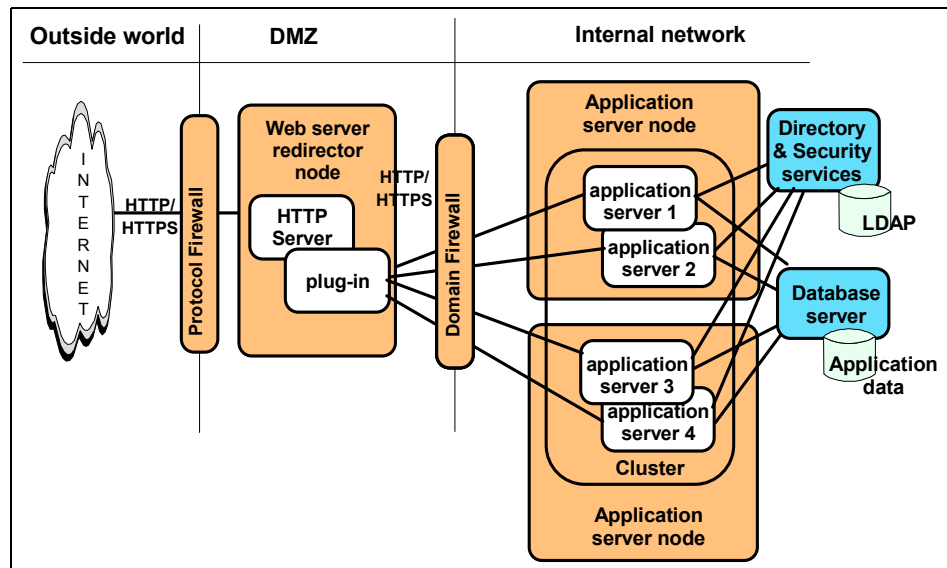


Figure 3-8 Horizontal scaling with cluster topology

The HTTP server distributes requests to cluster member application servers on the application server nodes.

The WebSphere Load Balancer component of IBM WebSphere Application Server Network Deployment V5.1, which can distribute client HTTP requests, can be combined with clustering to reap the benefits of both types of scaling. See 3.8.1, “Horizontal scaling with IP sprayer” on page 71 for a description of this configuration.

## **Advantages**

Horizontal scaling using clusters has the following advantages:

- ▶ **Availability**  
Provides the increased throughput of vertical scaling topologies but also provides failover support. This topology allows handling of application server process failures and hardware failures without significant interruption to client service.
- ▶ **Throughput**  
Optimizes the distribution of client requests through mechanisms such as workload management or remote HTTP transport.

## **Disadvantages**

Horizontal scaling using clusters has the following disadvantages:

- ▶ **Maintainability**  
With the concept of cells and clusters in IBM WebSphere Application Server Network Deployment V5.1, it is easy to administer multiple application servers from a single point. However, there is more installation and maintenance associated with the additional machines.
- ▶ **Cost: more machines.**

### **3.8.1 Horizontal scaling with IP sprayer**

Load-balancing products can be used to distribute HTTP requests among application server instances that are running on multiple physical machines.

The Load Balancer product that is part of the IBM WebSphere Application Server Network Deployment V5.1 Edge Components is an IP sprayer that performs intelligent load balancing among Web servers using server availability, capability, workload, and other criteria.

## Simple IP sprayer topology

Figure 3-9 depicts a simple horizontal scaling configuration that uses an IP sprayer on the load balancer node to distribute requests among application servers on multiple machines.

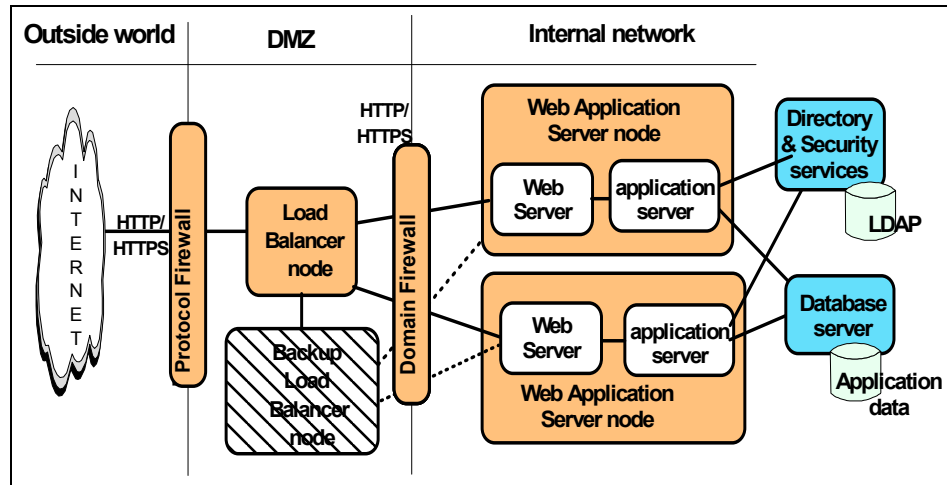


Figure 3-9 Simple IP sprayer-based horizontally scaled topology

A backup node for the load balancer node is normally configured in order to eliminate it as a single point of failure.

**Note:** In an IP sprayer (or similarly configured) topology, each Web server can be configured to perform workload management over all application servers in a specific cluster. In this configuration, session affinity will be preserved, no matter which Web server is passed the request by the IP sprayer.

## Complex IP sprayer topology

Figure 3-10 on page 73 depicts a topology where an IP sprayer distributes requests among several machines containing Web servers and clustered application servers.



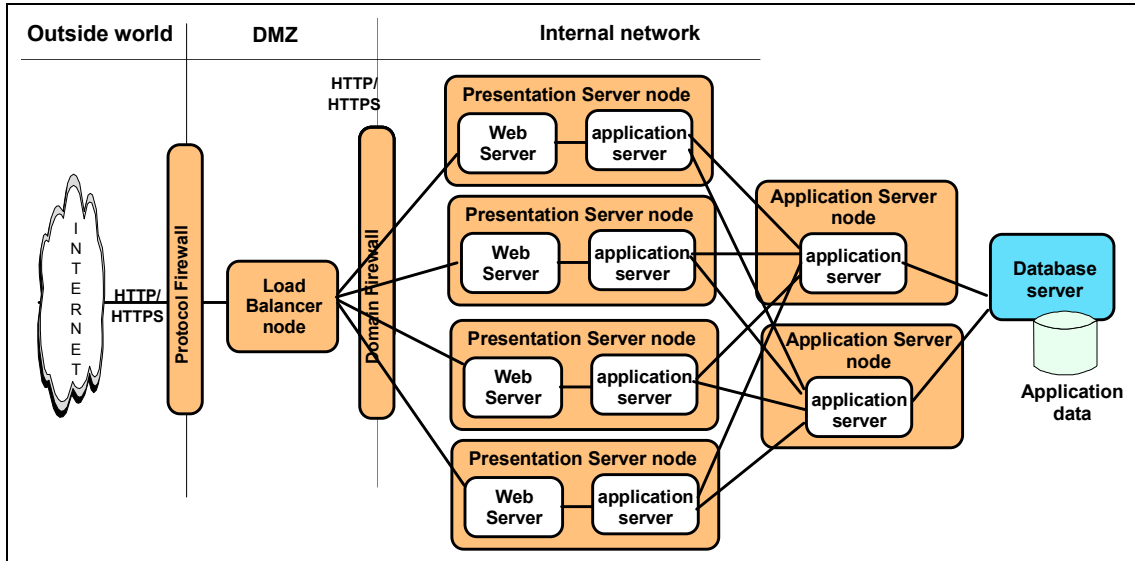


Figure 3-10 Complex IP sprayer horizontally scaled topology

The Presentation Server nodes host servlet-based applications. The Application Server nodes contain enterprise beans that access application data and execute business logic. This allows numerous less powerful machines to be employed in the first tier and fewer but more powerful machines in the second tier.

## Advantages

Using an IP sprayer to distribute HTTP requests has the following advantages:

- ▶ Improved server performance by distributing incoming TCP/IP requests (in this case, HTTP requests) among a group of Web server machines.
- ▶ The use of multiple Web servers increases the number of connected users.
- ▶ Elimination of the Web server as a single point of failure. It can also be used in combination with WebSphere workload management to eliminate the application server as a single point of failure.
- ▶ Improved throughput by letting multiple servers and CPUs handle the client workload.

## Disadvantages

Using an IP sprayer to distribute HTTP requests has the following disadvantage:

- ▶ Extra hardware and software is required for the IP sprayer server(s).

## 3.9 One WebSphere administrative cell versus many

Figure 3-11 depicts how an application can be implemented over multiple WebSphere Application Server Network Deployment cells.

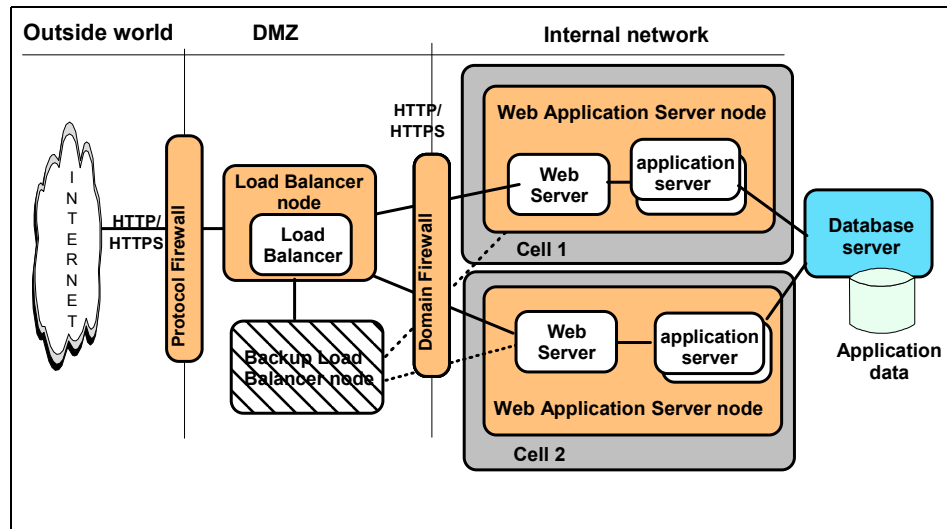


Figure 3-11 Multiple WebSphere cell topology

The example application runs simultaneously in two cells, each hosted on a different physical machine. WebSphere Load Balancer is used as an IP sprayer to distribute incoming HTTP requests among the two cells, presenting a single image of the application to the clients.

Each cell is administered independently, since each cell has its own set of XML configuration files.

**Tip:** A different version of the application can be run in each cell. In addition, because the cells are isolated from one another, different versions of the WebSphere Application Server software can be used in each cell.

There are no hard limits on the number of nodes and application servers that can be used in a single cell. However, the choice of whether to use a single or multiple cells can be made by weighing the advantages and disadvantages.

## Advantages

Using multiple cells has the following advantages:

- ▶ Isolation of hardware failure  
If one cell goes offline due to hardware problems, the others can still process client requests.
- ▶ Isolation of software failure  
Running an application in two or more cells isolates any problems that occur within a cell, while the other cells continue to handle client requests. This can be helpful in a variety of situations:
  - When rolling out a new application or a revision of an existing application. The new application or revision can be brought online in one cell and tested in a live situation while the other cells continue to handle client requests with the production version of the application.
  - When deploying a new version of the IBM WebSphere Application Server software. The new version can be brought into production and tested in a live situation without interrupting service.
  - When applying fixes or patches to the IBM WebSphere Application Server software. Each cell can be taken offline and upgraded without interrupting the application.
- ▶ If an unforeseen problem occurs with new software, using multiple cells can prevent an outage of an entire site. A rollback to a previous software version can also be accomplished more quickly. Hardware and software upgrades can be handled on a cell-by-cell basis during off-peak hours.

## Disadvantages

Using multiple cells has the following disadvantages:

- ▶ Deployment is more complicated than for a single administrative cell. Using a distributed file system that provides a common file mount point can make this task easier.
- ▶ Multiple cells require more administration effort because each cell is administered independently. This problem can be reduced by using scripts to standardize and automate common administrative tasks. In fact the use of scripted administration is likely essential in insuring repeatable processes that are required for a production environment.

### 3.10 Multiple clusters on one node versus one cluster per node

When deciding how to deploy an application, one decision to be made is whether to deploy a cluster of application servers across all machines, as illustrated in Figure 3-12, or only on a single machine, as illustrated in Figure 3-13 on page 77.

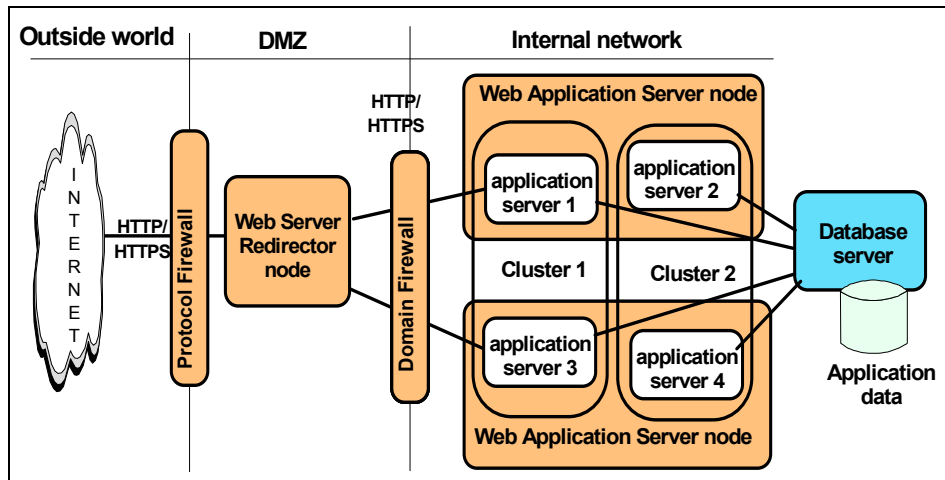


Figure 3-12 Multiple application server clusters on each machine

The topology in Figure 3-12 uses horizontal scaling. Each cluster of application servers is distributed throughout all of the machines in the system. In this example, a member of each cluster is hosted on each Web application server node. The Web server redirector node distributes client requests to the application servers on each node.

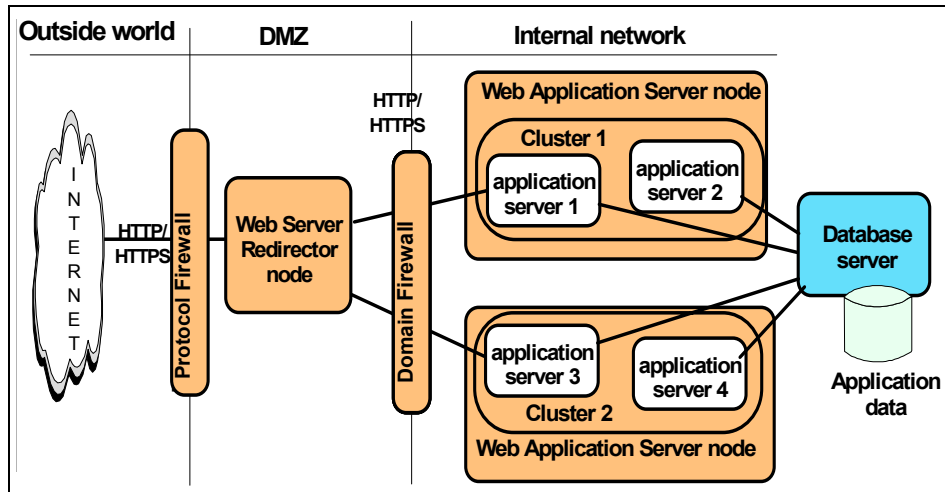


Figure 3-13 Single application server cluster on each machine

The topology in Figure 3-13 uses vertical scaling. Each cluster of application servers is located on a single machine of the system.

## Advantages

Hosting members of multiple clusters across one or more machines has the following advantages:

- ▶ **Improved throughput**  
Using an application server cluster enables the handling of more client requests simultaneously.
- ▶ **Improved performance**  
Hosting cluster members on multiple machines enables each member to make use of the machine's processing resources.
- ▶ **Hardware failover**  
Hosting cluster members on multiple nodes isolates hardware failures and provides failover support. Client requests can be redirected to the application server members on other nodes if a node goes offline.
- ▶ **Application software failover**  
Hosting cluster members on multiple nodes also isolates application software failures and provides failover support if an application server stops running. Client requests can be redirected to cluster members on other nodes.

- ▶ Process isolation

If one application server process fails, the cluster members on the other nodes are unaffected.

### **Disadvantages**

Hosting members of multiple clusters across one or more machines has the following disadvantage:

- ▶ Maintainability: Application servers must be maintained on multiple machines.
- ▶ Cost: more machines.

## **3.11 The sample topology**

We started our discussion on topologies with a single server configuration where all components reside on the same machine. Now we introduce a sample configuration that explores the subjects discussed so far:

- ▶ Separating the HTTP server
- ▶ Separating the database
- ▶ Vertical scaling
- ▶ Horizontal scaling
- ▶ HTTP server clustering

To configure, test, and evaluate the behavior of several components, we set up the configuration illustrated in Figure 3-14 on page 79 and used it throughout this book to test and evaluate aspects of IBM WebSphere Application Server V5.1 scalability. Chapter 7, “Implementing the sample topology” on page 255, describes the steps to set up this sample configuration.

The resulting topology is composed of:

- ▶ A cluster of two Web servers (IBM HTTP Server) anticipated by a Caching Proxy and a Load Balancer (both from WebSphere Edge Components).
- ▶ A dedicated Deployment Manager machine managing the WebSphere Application Server cell, running IBM WebSphere Application Server Network Deployment V5.1.
- ▶ A WebSphere cluster of three application server processes on two physical machines. WebSphere Application Server V5.1 is installed on both machines.
- ▶ A dedicated database server running IBM DB2 UDB V8.1.

**Important:** To make it easier to demonstrate the workload management and failover capabilities of WebSphere we have chosen to split the Web container and the EJB container in our sample topology. As mentioned earlier, while this is a possible configuration, however, it is not recommended because doing so will most likely have a negative impact on the performance. This as a result of the out of process calls from the EJB clients in the Web container to the EJB container.

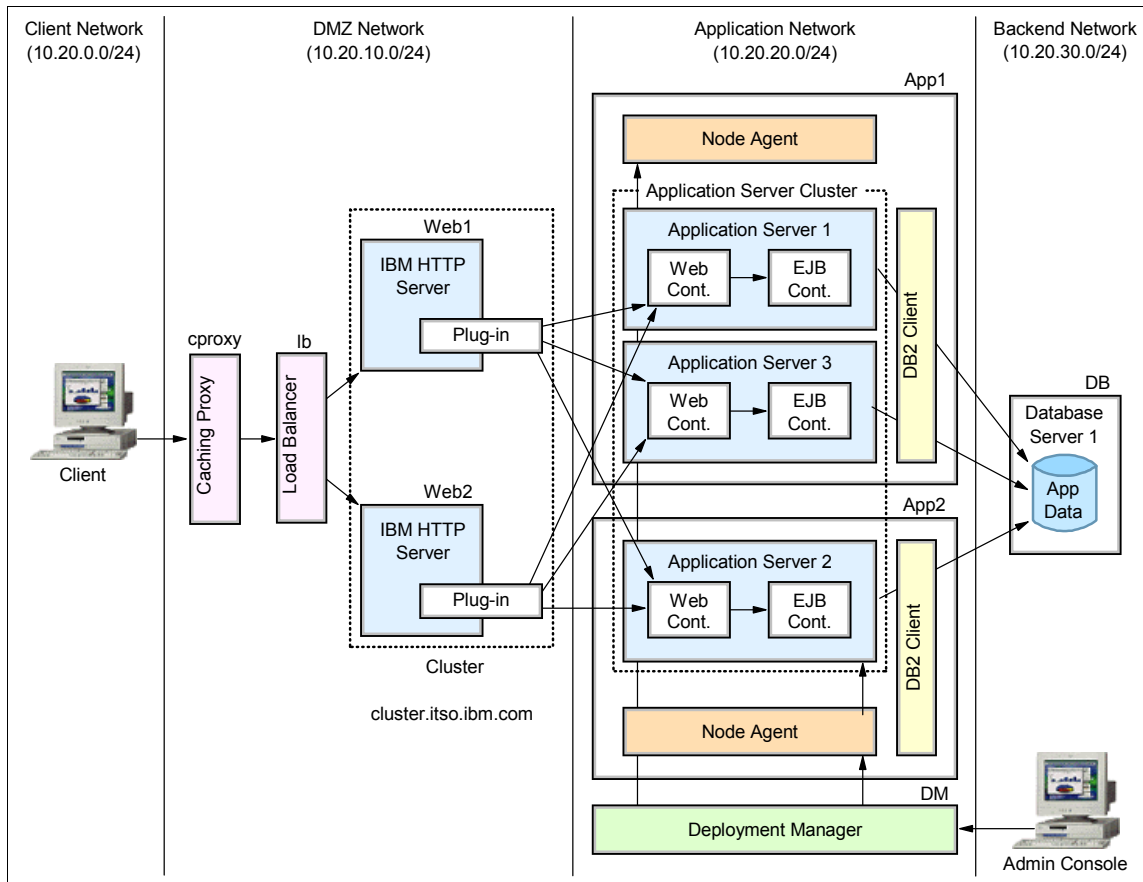


Figure 3-14 ITSO sample topology

We have two HTTP servers configured as a cluster under the control of WebSphere Load Balancer. The HTTP servers are configured to respond both to port 80 (HTTP) and port 443 (HTTPS). The plug-in configuration file (plugin-cfg.xml) must be copied to both servers. The Load Balancer server distributes requests among the HTTP servers based on weights set by the Load Balancer's manager component.

As well as monitoring the network reachability of the machines, an HTTP advisor is activated to poll the availability of the HTTP service on each machine.

## **3.12 Topologies and high availability**

In order to select the most appropriate topology, it is important to understand how WebSphere provides high availability for the Web container and EJB container. Therefore, we examine the three failure situations shown in Figure 3-15 on page 81: HTTP server, Web container, and EJB container.



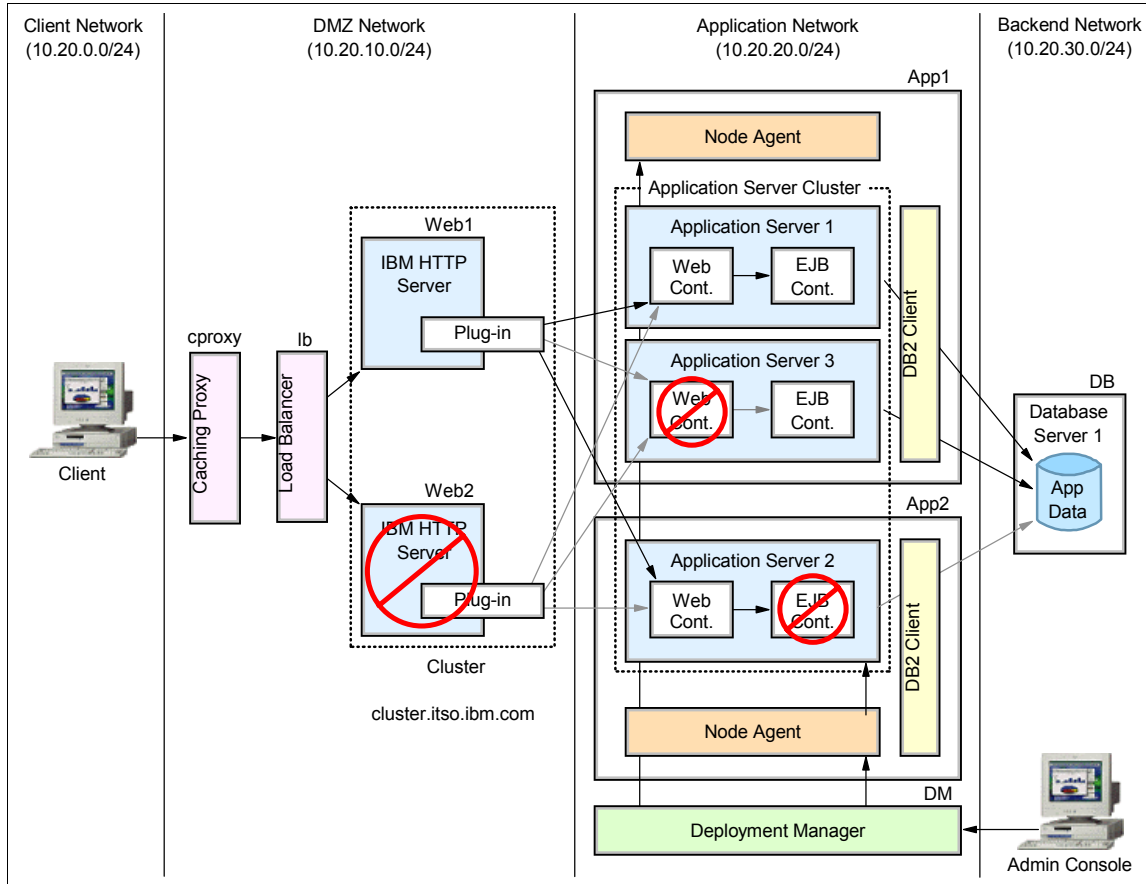


Figure 3-15 Failure situations

In normal operation, the HTTP requests are forwarded to one of the clustered machines (web1 or web2) acting as HTTP servers. If the requests should be served by WebSphere, the plug-in uses HTTP transport to route requests to the application servers on system app1 or app2. The Web applications are deployed on member application servers in the WEBcluster (web1a and web1b on app1, and web2 on app2).

In case of a failure of one of these application servers, or if the application server is not active, the plug-in redirects the requests to the available members in the same cluster (WEBcluster). The requests will be sent using HTTP, either to the same machine or to a separate machine. For plug-in workload management and failover details, refer to Chapter 5, "Plug-in workload management and failover" on page 133.

The same applies to EJB containers. We configured a cluster consisting of applications servers in EJBcluster (EJB1a and EJB1b on system app1, and EJB2 on app2). If an EJB application server is not available, requests are directed to the available EJB application server in the EJBcluster. Chapter 6, “EJB workload management” on page 215 provides further details on EJB workload management.

If an HTTP server fails, Load Balancer redirects HTTP requests to the available HTTP servers on the cluster. Figure 3-16 on page 83 shows how requests are routed around the failed Web server and application servers in our sample topology.

**Note:** Failure of Deployment Manager does not interrupt the managed servers. The Deployment Manager node stores the master configuration repository and every managed server stores only the appropriate subset of the master configuration. Therefore, we do recommend maintaining the master configuration repository on a shared file server. The objective is to recover the configuration data, in exceptional cases such as a complete failure of the machine running Deployment Manager. See the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195 for more information about the configuration repository.

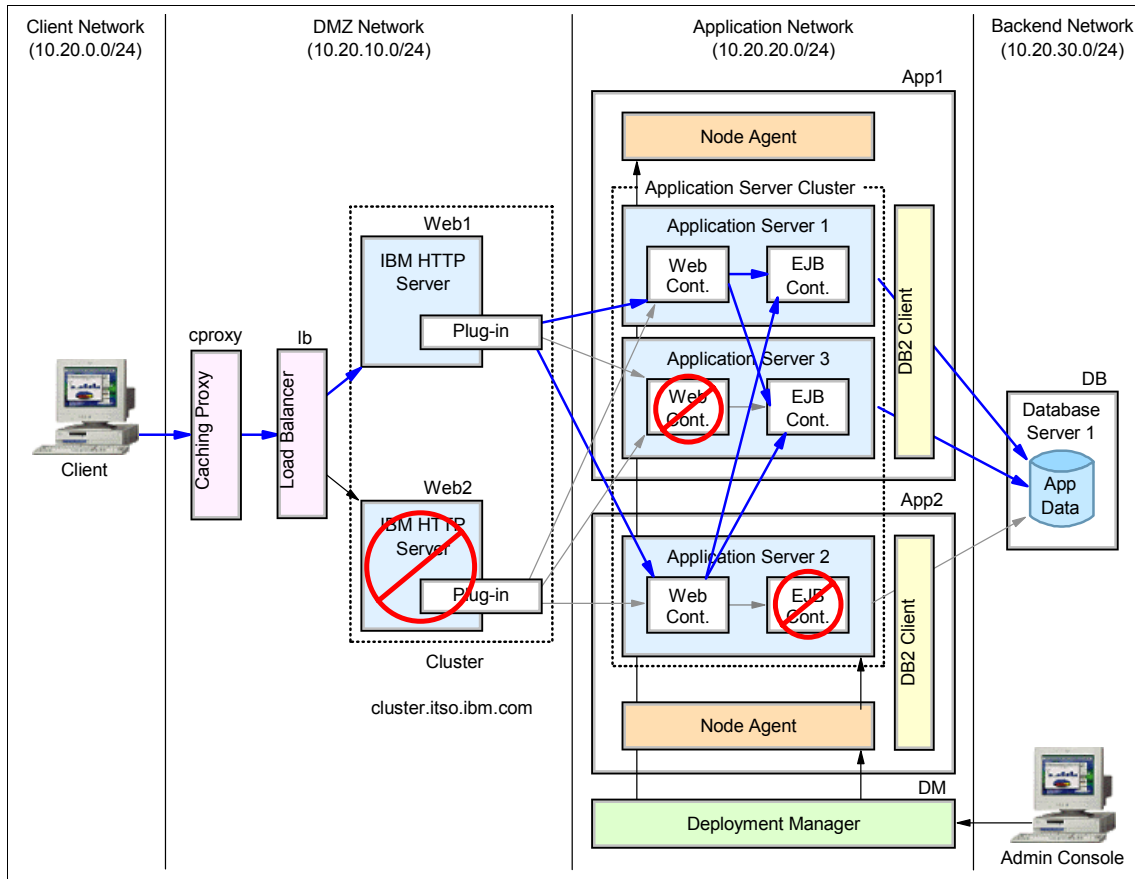


Figure 3-16 Failover behavior

### 3.12.1 Using WebSphere Load Balancer custom advisor

If the WebSphere machine has stopped or the application server is down while the HTTP server is still running, you will receive an error message on the browser, as illustrated in Figure 3-17 on page 84.

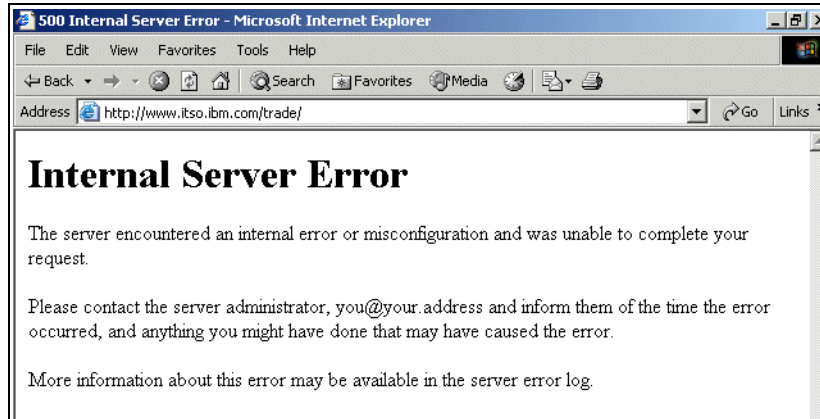


Figure 3-17 Message when application server is down

To help us avoid this problem, WebSphere Load Balancer provides a sample custom advisor for WebSphere Application Server. We used this advisor instead of the default HTTP advisor. The layout of our solution is shown in Figure 3-18.

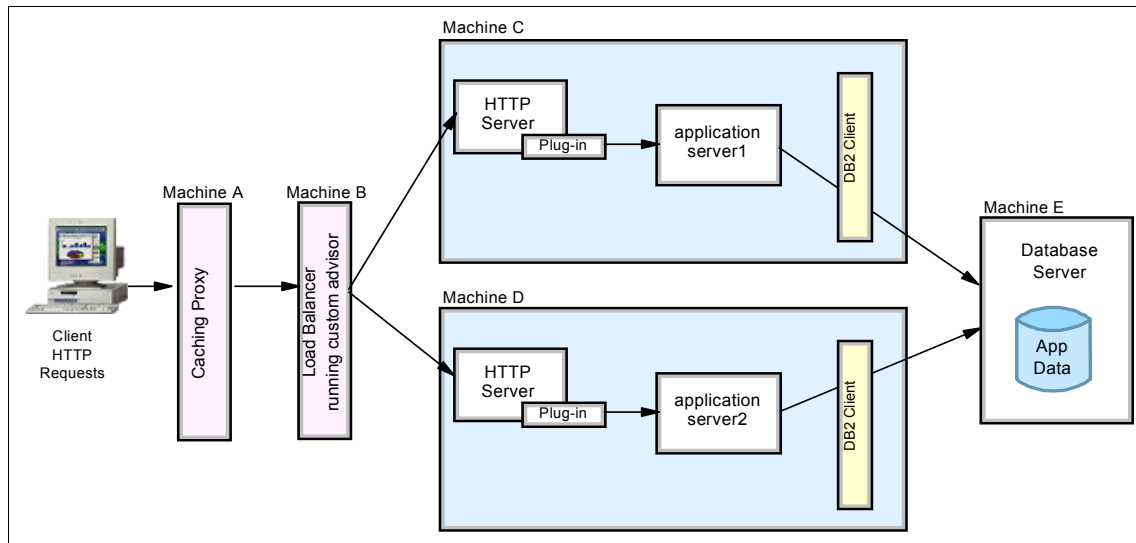


Figure 3-18 Using WebSphere custom advisor

Now, when the WebSphere custom advisor is running, we can continue to access the application server2 on machine D without getting the error message even when machine C is stopped. The custom advisor directs the request to the HTTP server on machine D, as shown in Figure 3-19 on page 85.

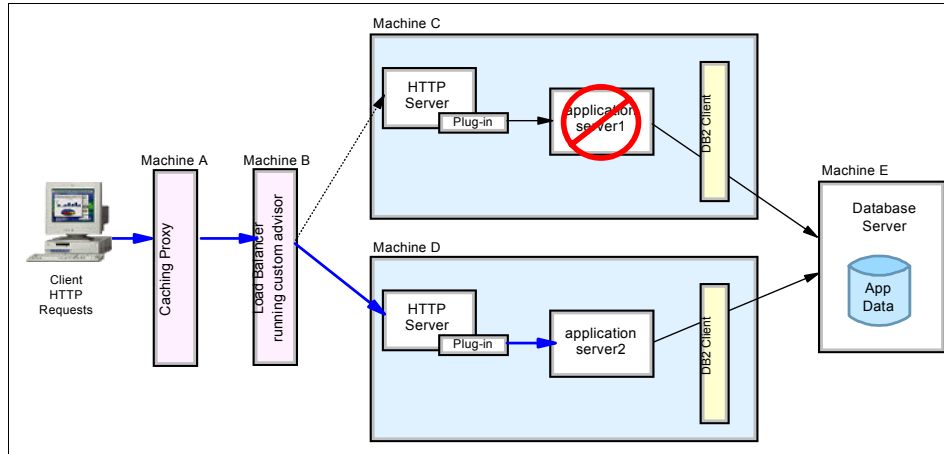


Figure 3-19 Requests dispatched to active application server

#### Notes:

- ▶ You cannot run the HTTP advisor and the WebSphere custom advisor at the same time if you specify the same port number for both advisors.
- ▶ WebSphere custom advisor must be considered as a monitoring extension associated with each HTTP server on the cluster. The custom advisor prevents requests from being sent to a specific HTTP server when this HTTP server cannot appropriately fulfill them (for example, when the WebSphere server it sends requests to is down).

When we use WebSphere workload management, the requests from multiple HTTP servers are sent to a group of application servers, distributed also among multiple machines. We cannot associate the service layer provided by the application server to an HTTP server anymore, since the plug-in is responsible for distributing the requests.

WebSphere custom advisor can have a more meaningful use when WebSphere workload management is not used or a whole cluster associated to a specific HTTP server is down.

When using WebSphere plug-in workload management, as in our sample topology, monitoring the HTTP servers using the HTTP advisor is probably the best choice.

## 3.13 Closing thoughts on topologies

Regardless of which topology you decide to use, a best practice is to partition your production acceptance environment in exactly the same way as your production environment. This avoids surprises when deploying your application into production.

Another consideration, when practical for your application architecture, is to create a number of smaller application servers, rather than a single large one. This has at least two advantages:

- ▶ The plug-in configuration files can be smaller (less complexity of URIs), which leads to better startup performance and possibly to better execution performance.
- ▶ At least during the development phase, it takes less time to cycle a smaller application server to pick up various configuration changes.

Of course, the benefits of creating multiple application servers in this manner must be carefully balanced against the increased complexity of doing so, and the potential increase in response time due to interprocess RMI/IIOP calls and network latency.

If, after looking at all the topologies, you are still wondering what would be the best possible topology, one possibility is illustrated in Figure 3-20 on page 87.

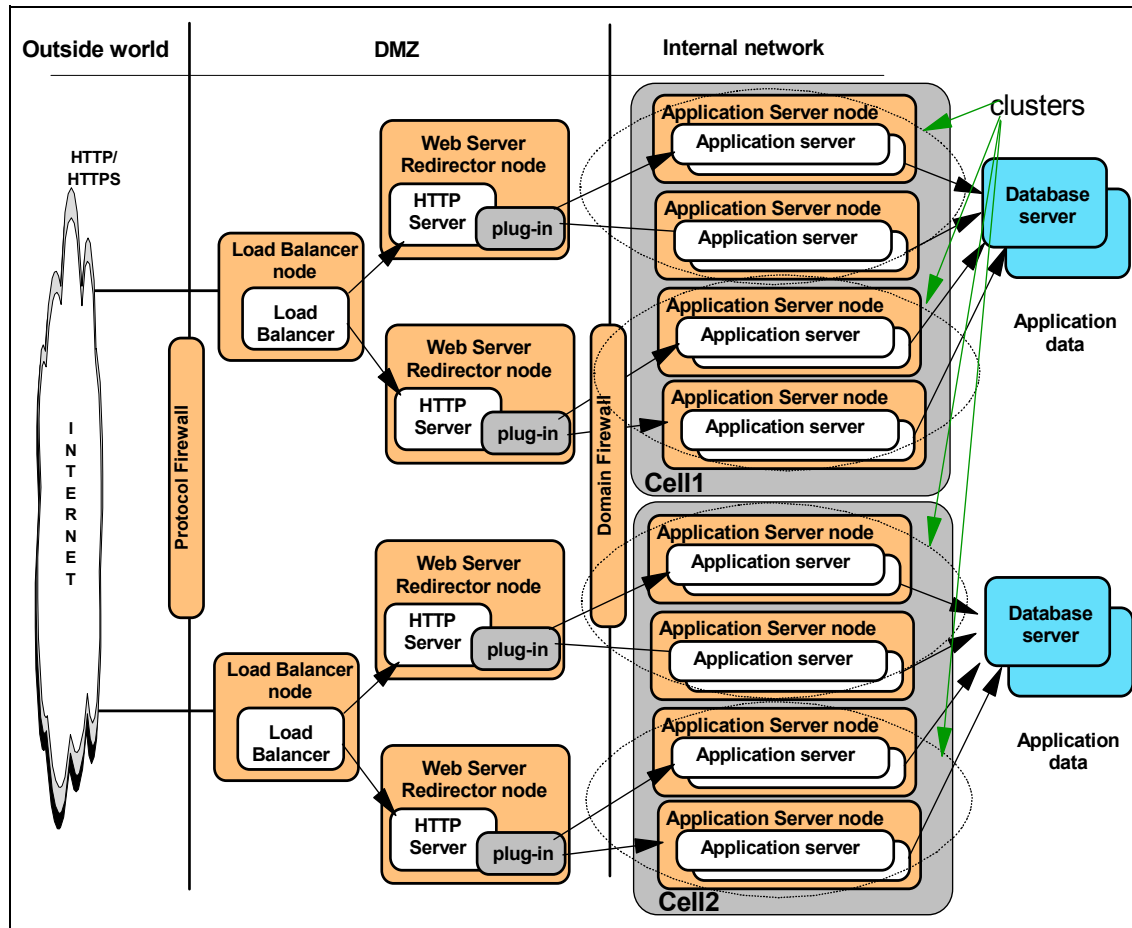


Figure 3-20 One of the best possible topologies

This topology includes two of everything:

- ▶ Cells
- ▶ Database servers (two for each cell - you need to ensure data replication between these database servers)
- ▶ Application servers (two sets for each cluster)
- ▶ HTTP servers (two for each cell)
- ▶ HTTP dispatchers (WebSphere Load Balancer or a similar product)

Only the loss of an entire cell will normally be noticed by users. If this were to occur, the active HTTP sessions would be lost for half of the users, requiring that they log in again. Of course, performance would most likely be degraded in this

case as well, but it would ensure continuous availability. In most cases, hardware or software upgrades would be handled on a cell-by-cell basis during non-peak hours. If customer satisfaction (performance, throughput, availability) is important for your Web site, then this would be the appropriate topology.

## 3.14 Topology selection summary

Table 3-1 provides a summary of some considerations for topology selection as discussed in this chapter.

*Table 3-1 Topology selection considerations*

	<b>Security</b>	<b>Performance</b>	<b>Throughput</b>	<b>Maintain-ability</b>	<b>Availability</b>	<b>Session</b>
Single machine	Little isolation between components	Competition for machine resources	Limited to machine resources	Ease of installation and maintenance	Machine is single point of failure	
Remote Web server	Allows for firewall/DMZ	<ul style="list-style-type: none"> <li>- Separation of loads</li> <li>- Performance usually better than local</li> </ul>	Independent tuning	<ul style="list-style-type: none"> <li>- Independent configuration and component replacement</li> <li>- More administrative overhead</li> <li>- Need copy of plug-in config.xml file</li> </ul>	Introduces single point of failure	
Separate database server	Firewall can provide isolation	Separation of loads	<ul style="list-style-type: none"> <li>- Independent tuning</li> <li>- Must consider network bottleneck</li> </ul>	<ul style="list-style-type: none"> <li>- Use already established DBA procedures</li> <li>- Independent configuration</li> <li>- More administrative overhead</li> </ul>	<ul style="list-style-type: none"> <li>- Introduces single point of failure</li> <li>- Use already established HA servers</li> </ul>	
Separate Web/EJB container	More options for firewalls	Typically slower than single JVM	Clustering can improve throughput	More administrative overhead	Introduces single point of failure	



	Security	Performance	Throughput	Maintainability	Availability	Session
Vertical scaling		Improved throughput on large SMP servers	Limited to resources on a single machine	Easiest to maintain	<ul style="list-style-type: none"> <li>- Process isolation</li> <li>- Process redundancy</li> </ul>	<ul style="list-style-type: none"> <li>- May use session affinity</li> <li>- Memory-to-memory session replication or persistent session database for session failover</li> </ul>
Horizontal scaling		Distribution of load	Distribution of connections	More to install / maintain	Process and hardware redundancy	<ul style="list-style-type: none"> <li>- May use session affinity</li> <li>- Memory-to-memory session replication or persistent session database required for session failover</li> </ul>
Add Web server		Distribution of load	Distribution of connections	<ul style="list-style-type: none"> <li>- More to install / maintain</li> <li>- Need replication of configs / pages</li> </ul>	Best in general	Use load balancer SSL session id affinity when using SSL
One cell				Ease of maintenance		
Multiple cells		Less lookups and interprocess communication		Harder to maintain than single cell	Process, hardware and software redundancy	





## Part 2

# Distributing the workload





## Web server load balancing

In this chapter, we describe Web server load balancing and proxy techniques, using the IBM WebSphere Edge Components available with IBM WebSphere Application Server Network Deployment V5.1. We cover the Load Balancer (previously known as Network Dispatcher) and Caching Proxy (previously known as Web Traffic Express).

## 4.1 Introduction

In a real-world environment, when the number of users accessing your Web site increases, you would experience slow response times or connections being refused. Your site may even fail under critical load conditions. Simply increasing the processing power and other resources on your server may not be cost effective anymore. You need to provide scalability for your environment and ensure that it has the required availability and provides better performance.

### 4.1.1 Scalability

Applications need to scale for increasing numbers of simultaneous users on a wide range of Web access devices.

By adding one or more Web servers to the existing one, you can prevent a single Web server from becoming overloaded. The incoming requests are then dispatched to a group of servers, called a cluster. A cluster is a group of independent nodes interconnected and working together as a single system.

A load balancer software is used to dispatch the load to the Web servers in the cluster. It uses a load-balancing mechanism usually known as *IP spraying*, which intercepts the HTTP requests and redirects them to the appropriate machine in the cluster, providing scalability, load balancing, and failover.

### 4.1.2 Availability

Users must be able to reach the application regardless of failed server sites. In a clustered Web server environment, the load balancer monitors the availability of the Web servers. If a Web server has failed, no more requests are sent to the failed system. Instead, all requests are routed to other, active Web servers. It is also recommended that you ensure high availability of the load balancer system itself to eliminate it as a single point of failure.

### 4.1.3 Performance

Quick response times can be provided by routing requests based on the geographic location, user identity, or content requested and by caching the retrieved data.

## 4.2 IBM WebSphere Edge Components

The IBM WebSphere Edge Components, which are part of IBM WebSphere Application Server Network Deployment V5.1, help you to reduce Web server congestion, increase content availability, and improve Web server performance.

The following functions are included in the Edge Components of IBM WebSphere Application Server Network Deployment V5.1:

- Load Balancer
- Caching Proxy

**Note:** WebSphere versions before V5.0 did *not* include these functions. Instead you had to separately purchase a load balancer and/or caching proxy software. The appropriate IBM product is called IBM WebSphere Edge Server. This software is still available as a stand-alone product with additional functions, such as Transactional Quality of Service and Content Distribution. However, if you need only the Edge Component functions of Load Balancer and Caching Proxy, you do not need to purchase IBM WebSphere Edge Server separately.

WebSphere Edge Components are supported on a wide range of operating systems, such as AIX, Linux, Solaris, and Windows 2000.

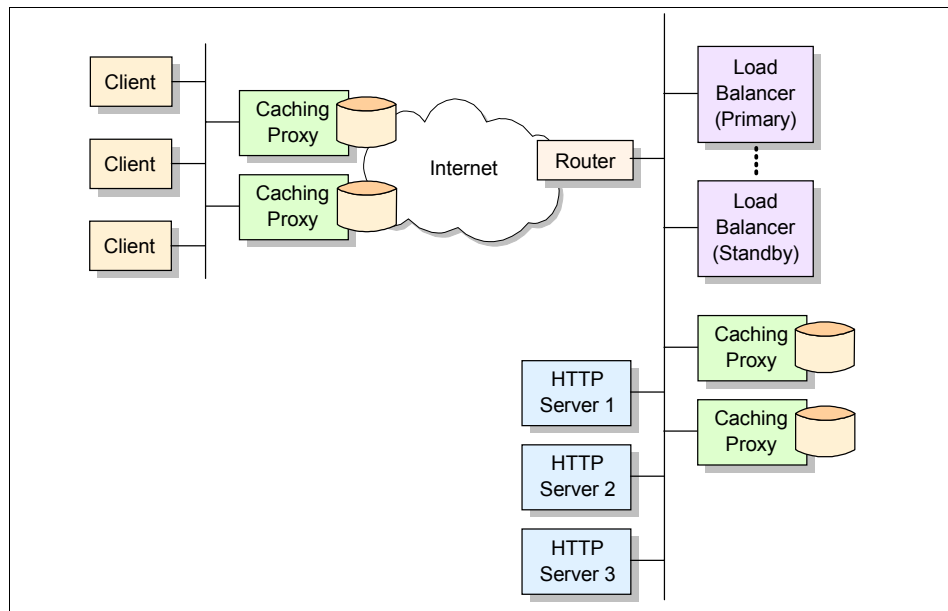


Figure 4-1 Edge Components

**Note:** In IBM WebSphere Edge Server, the Load Balancer component was referred to as Network Dispatcher, and Caching Proxy was referred to as Web Traffic Express.

The Load Balancer is covered in detail in 4.3, “Load Balancer overview” on page 96, and the Caching Proxy in 4.6, “Caching Proxy” on page 127.

## 4.3 Load Balancer overview

The Load Balancer is responsible for load balancing traffic to servers that are in a LAN or WAN. Load Balancer consists of the following five components that can be used separately or together:

- ▶ Dispatcher component
- ▶ Content Based Routing (CBR) for HTTP and HTTPS
- ▶ Site Selector
- ▶ Cisco CSS Controlller
- ▶ Nortel Alteon Controller

### Dispatcher component

The Dispatcher component performs intelligent load balancing by using server availability, capability, workload, and other criteria you can define, to determine where to send a request. These weights and measurements are set dynamically by the Dispatcher.

Load balancing is provided at a specific service level, such as HTTP, FTP, SSL, NNTP, IMAP, POP3, SMTP, and Telnet. It does not use a domain name server to map domain names to IP addresses and it supports such forwarding methods as Media Access Control (MAC), Network Address Translation (NAT), Network Address Port Translation (NAPT), and CBR Forwarding.

The core function of Dispatcher is the *Executor*. It is a kernel-level function (or device driver) that examines only the header of each packet and decides whether the packet belongs to an existing connection, or represents a new connection request. The Executor uses a connection table in memory to achieve this. Note that the actual connection is never set up on the Dispatcher machine. The connection is between the client and server, just as it would be if the Dispatcher were not installed. But the connection table records its existence and the address of the server to which the connection was sent.

Based on information provided by the *Manager*, the Executor selects to which server it will forward a new connection packet. The Manager periodically sets weights for the servers and may use *Advisors*, which are lightweight clients that



run inside the Dispatcher. Standard advisors execute a trivial command to determine the status of the servers. If the request succeeds, the server is considered up. A server that is very busy (or slower in responding to an advisor request) will receive a higher port load and lower overall weight than a server that has more capacity available and fewer new requests will be routed to the busy server. A down server is given a weight of zero, and packets will not be forwarded to it until the server responds to the advisor. A server that was manually marked down has a weight of -1.

You can also use *Metric server*, a system monitoring component of Load Balancer that must be installed in each load-balanced server within your configuration. Metric server can then provide customizable monitoring of the activity on a server, contributing to the weight value for the server. The script for Metric server must return a value from 0-100 (or -1 if you want to mark the server down) to define the weight for each server. In other words, one weight will be recorded for all ports configured to that server. This will be reported in the manager report under the system column.

The interaction of Executor, Manager, and other Load Balancer components are shown in Figure 4-2.

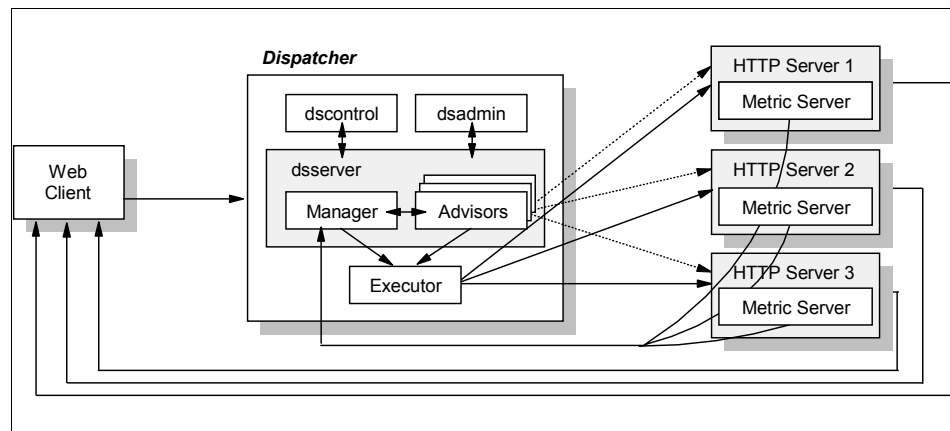


Figure 4-2 Dispatcher components interaction

The command-line interface for Dispatcher is *dscontrol*. Use *lbadmin* to start the GUI interface.

## Content Based Routing (CBR)

The CBR component makes load balancing based on the content of the request. Load Balancer supports content-based routing in two ways: the CBR component and the Dispatcher component's cbr forwarding method.

In conjunction with Caching Proxy, the CBR component has the ability to proxy HTTP and HTTPS (SSL) requests to specific servers based on the content requested. The Dispatcher component also provides content-based routing, but it does not require the Caching Proxy to be installed. Because the Dispatcher component's content-based routing is performed in the kernel as packets are received, it can provide faster content-based routing than the CBR component.

When do you use which CBR method?

- ▶ For fully secure SSL traffic (client through server):
  - The CBR component (in conjunction with Caching Proxy) can process SSL encryption/decryption in order to perform content-based routing.
  - The Dispatcher's cbr forwarding can only be configured with SSL ID affinity because it cannot process the encryption/decryption to perform true content-based routing on the client request's URL.  
  
Refer to 4.5.5, "SSL session ID" on page 120 for configuration details.
- ▶ For HTTP traffic:
  - The Dispatcher's cbr forwarding method provides a faster response to client requests than the CBR component. Also, Dispatcher's cbr forwarding does not require the installation and use of Caching Proxy.

## Site Selector

This component performs load balancing using a DNS round robin approach or a more advanced user-specified approach. Site Selector works in conjunction with a name server to map DNS names to IP addresses. System Metrics (provided by the metric server) should be used in addition to advisor weights to achieve a well balanced/accurate weighting of servers.

## Cisco CSS Controller and Nortel Alteon Controller

These controllers can be used to generate server weighting metrics that are then sent to the Cisco and Alteon Switch respectively for optimal server selection, load optimization, and fault tolerance.

### 4.3.1 Load Balancer topologies

Load Balancer can be configured for different topologies, depending on the number of machines available and the high availability requirements. In this section we discuss the following topologies:

- ▶ Load Balancer on a dedicated server
- ▶ Collocated servers
- ▶ High availability
- ▶ Mutual high availability

#### Load Balancer on a dedicated server

This is the basic topology where you install Load Balancer on a dedicated server and configure it to balance the workload between multiple Web servers as shown in Figure 4-3.

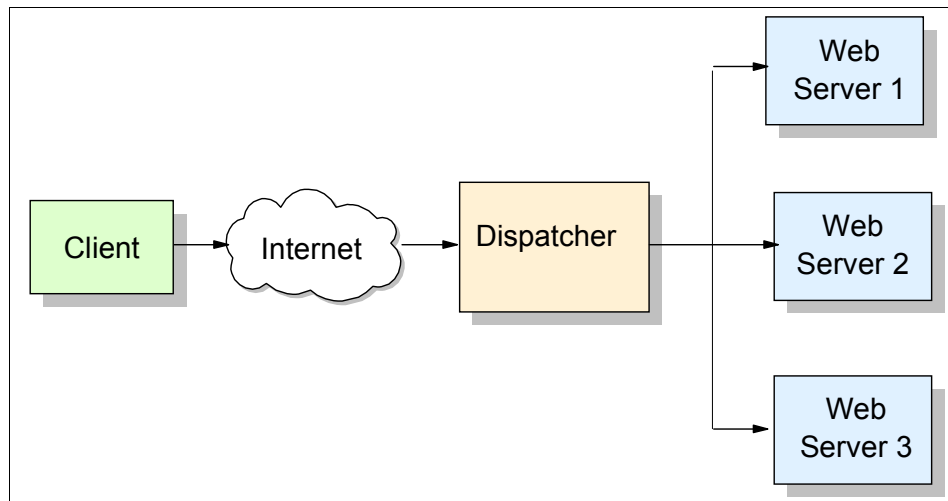


Figure 4-3 Dispatcher basic topology

#### Collocated servers

Load Balancer can reside on the same machine as a Web server to which it is distributing incoming requests. This is commonly referred to as *collocating* a server. Collocated servers keep initial costs down but allow evolving to a more complex configuration, such as a dedicated server for Dispatcher or even a high availability solution. Collocation is also supported for CBR, but only when using bind-specific Web servers and bind-specific Caching Proxy.

Collocation applies to the Dispatcher and Site Selector components.

In a Windows 2000 environment, you can collocate Dispatcher, when using the Dispatcher's NAT and cbr forwarding methods, but it is not supported when using Dispatcher's MAC forwarding method.

**Note:** To use collocated servers in a Linux environment, you must apply an arp patch which varies for the different kernel versions of Linux. For details on what you need to apply, refer to the Edge Components InfoCenter at:

<http://www.ibm.com/software/webservers/appserv/doc/v51/ec/infocenter/index.html>

Select the Load Balancer Administration Guide from the left hand pane. The Administration Guide is displayed in the right pane. Scroll down to the "Requirements for Linux" section. Here you find a link called "Installing the Linux kernel patch (to suppress arp responses on the loopback interface)" which contains detailed information for this issue.

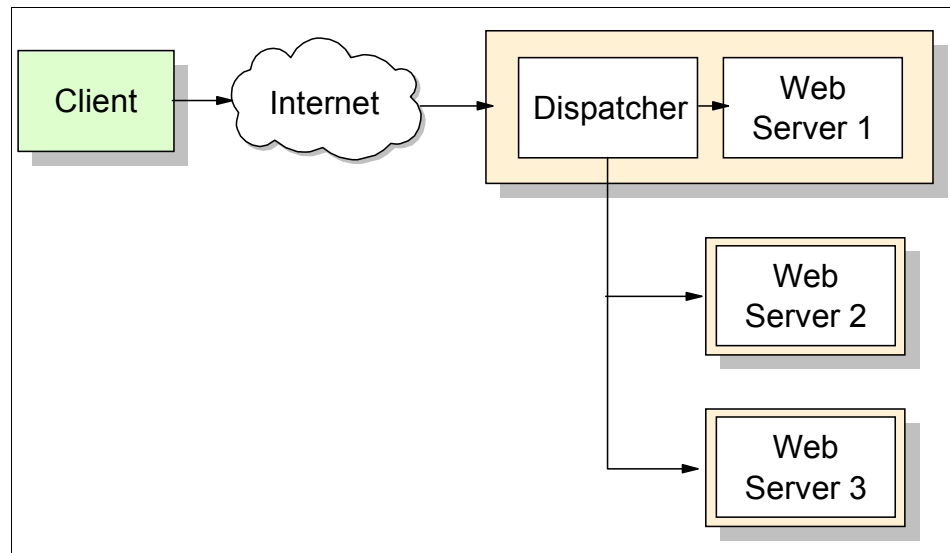


Figure 4-4 Dispatcher collocated topology

**Note:** A collocated Web server competes for resources with Load Balancer during times of high traffic. However in the absence of overloaded machines, using collocated servers offers a reduction in the total number of machines necessary to set up a load-balanced site.

## High availability

In order to eliminate the Dispatcher itself as a single point of failure, you can configure it for high availability by providing a secondary machine.

The first (active) Dispatcher machine performs load balancing in the same way as a single Dispatcher configuration. The second system monitors the health of the first, and takes over the load balancing if it detects that the first Dispatcher has failed. The two machines need to reside on the same subnet. This configuration is illustrated in Figure 4-5 on page 102.

Each of the two machines is assigned a specific role, either primary or backup. The primary machine constantly sends connection data to the backup system. Each machine is monitoring the health of the other through heartbeats. Heartbeats are sent every half second, the failover occurs when the backup machine receives no response from the primary within two seconds. Another reason for a failover is when the backup machine is able to ping more reach targets than the primary machine.

The failover process is as follows: The standby machine becomes active, takes over the cluster IP address, and assumes the role of forwarding packets. The primary machine automatically becomes standby.

**Note:** By default, high availability is only supported for the Dispatcher component, not for the Site Selector.

For a highly available Site Selector, you should configure Dispatcher to host a cluster in front of multiple Site Selectors. The cluster will act as the sitename DNS clients are requesting. The Dispatcher tier can provide the failover mechanism and the Site Selectors will then have no single point of failure. The DNS advisor to be used to detect a failed site selector is included with the product.

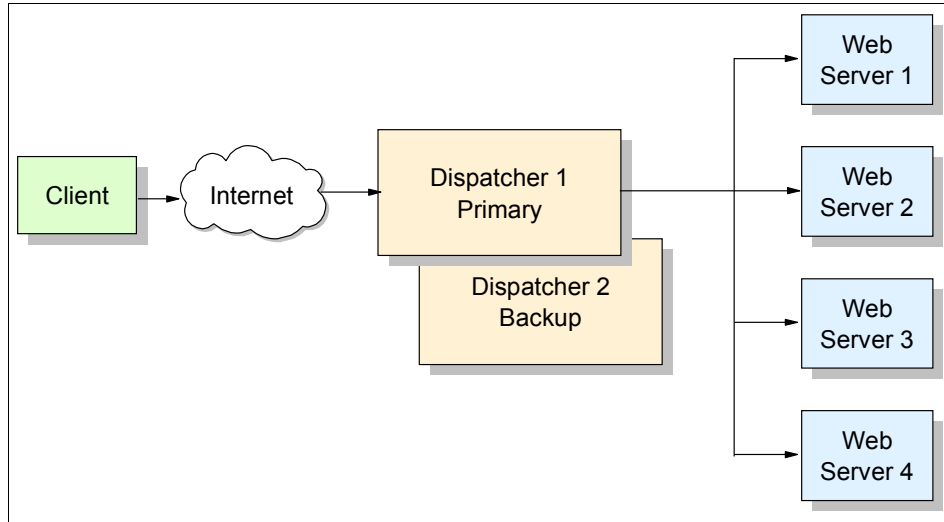


Figure 4-5 Dispatcher high-availability topology

## Mutual high availability

You have to configure two Dispatcher machines, both of which actively perform load balancing and provide backup for each other. In a simple high-availability configuration, only one machine performs load balancing. In mutual high availability, both machines participate in load balancing.

For mutual high availability, client traffic is assigned to each Dispatcher machine on a cluster address basis. As illustrated in Figure 4-6 on page 103, you have to configure a cluster for each node that is a primary Dispatcher (so you have at least two clusters in your environment). Each cluster can be configured with the NFA (non-forwarding address) of its primary Dispatcher. The primary Dispatcher machine normally performs load balancing for that cluster. In the event of failure, the other machine performs load balancing for both its own cluster and for the failed Dispatcher's cluster.

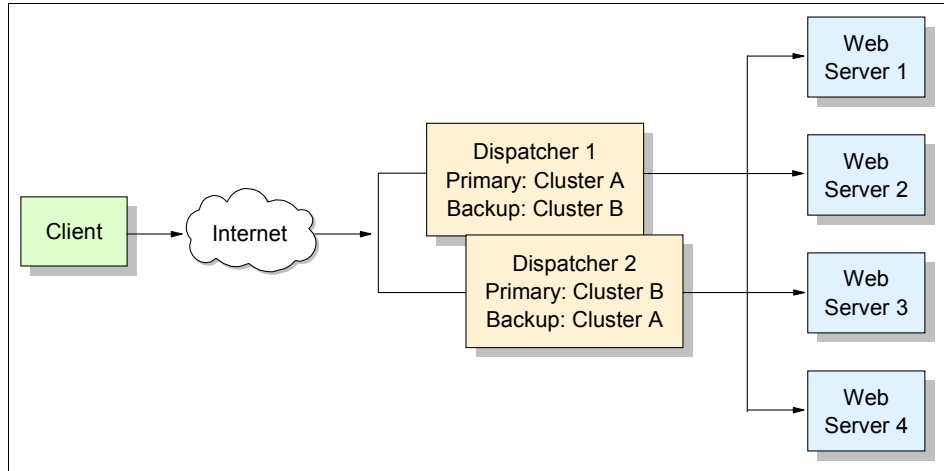


Figure 4-6 Dispatcher mutual high availability topology

## 4.3.2 Installation and configuration

First, select a suitable topology, depending on the requirements of your site. The detailed configuration of IBM Load Balancer is a complex process that is beyond the scope of this redbook. In particular, the procedures related to network settings may vary, depending on the operating system being used and its version. We will describe only the basic configuration steps used in our sample topology. For detailed information about installing the Load Balancer, refer to the *Load Balancer Administration Guide*, GC09-4602, available on the Internet at:

<http://www.ibm.com/software/webservers/appserv/doc/v50/ec/infocenter/index.html>

## 4.3.3 Setting up the cluster machines

This section introduces generic concepts and procedures related to setting up an HTTP server cluster using Load Balancer Dispatcher.

**Note:** Please note that the configuration described here is not using the same names as in our sample topology from Chapter 7, “Implementing the sample topology” on page 255.

The basic steps are:

1. Ensure that the Load Balancer server and the HTTP servers are on the same LAN segment.

2. Ensure that the Load Balancer server can ping the Web servers.
3. Ensure that each HTTP server is operational.
4. Allocate a network address for the cluster, such as `www.itso.ibm.com`. This is the address that the browser will point to in order to access the site (`http://www.itso.ibm.com`).
5. Save a map of the routing tables of the HTTP server machines for later reference. Use the commands **netstat -ar** on UNIX or **route print** on Windows 2000, redirecting the output to a file.
6. Add an alias to the loopback interface on the clustered HTTP servers. On AIX, use the command:  

```
ifconfig lo0 alias www.itso.ibm.com netmask 255.255.0.0
```
7. Check the routing table and delete any extra routes to your network. An AIX example would be:  

```
route delete -net <network_address> <cluster_address>
```
8. On each HTTP server, check that you can ping the alias address.

#### 4.3.4 Configuring a Web server cluster

Configuring a Web server cluster using the Load Balancer can be accomplished either through the command line or through a graphical user interface (GUI).

As an example, the GUI representation of the configuration used in our sample is illustrated in Figure 4-7.



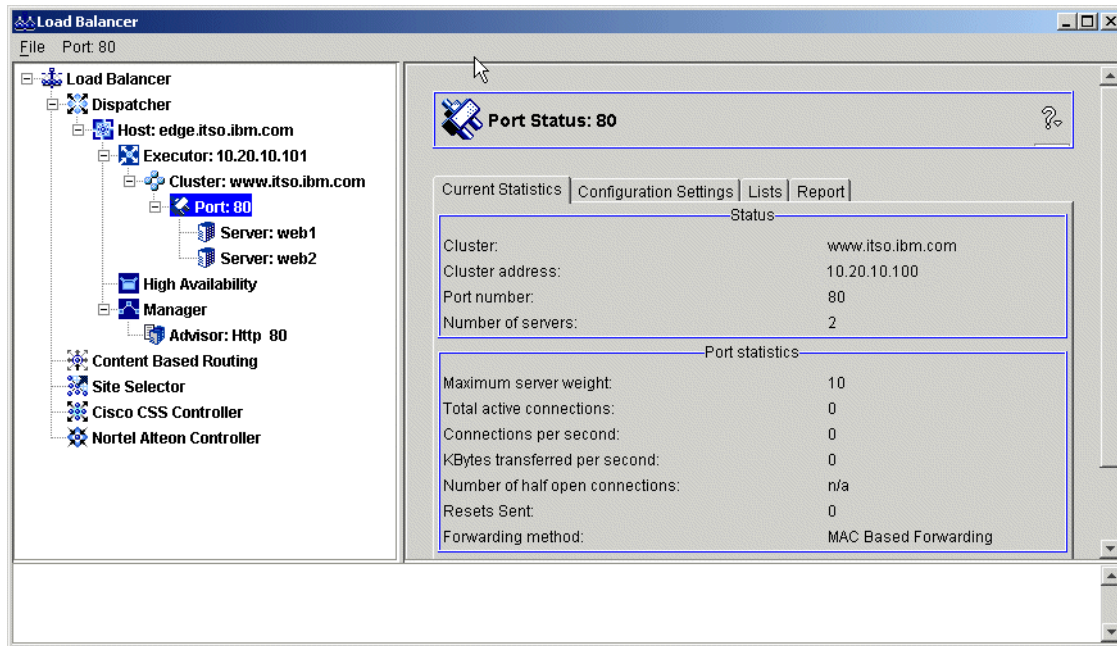


Figure 4-7 Load Balancer configuration

A GUI wizard is also available to step you through the process of creating a basic configuration for the Dispatcher component.

In addition, a command-line interface (**dscontrol** command) can be used, alone or as part of the script files, making it easier to perform startup procedures and automation tasks.

In this section, we describe the basic steps to set up our configuration of Load Balancer, using the **dscontrol** command. In 7.4, “Configuring Load Balancer” on page 264, we describe the basic configuration of the Web server clustering used in our sample topology, using **ladmin**, the administration GUI.

The configuration steps are:

1. Start the Load Balancer server

To start the server, type:

```
dsserver
```

Or,

```
dsserver start
```

This component is the basic user-space component of Load Balancer. Its role is to handle the command interface from the user, and to start and stop other components and threads. It is usually the first component started.

To stop the server, just type:

```
dsserver stop
```

## 2. Start the Executor function

To start the Executor function of the Dispatcher, which performs the core task of actually forwarding the packets, type:

```
dscontrol executor start
```

The Executor is the heart of the product, and the one kernel-space component of Load Balancer. Its main role is to forward packets. It receives packets from the network interface device, selects a server to handle the request, and forwards the packets to the selected server.

**Important:** The Executor will continue to run even when the Load Balancer server has been shut down.

Therefore, for a complete shutdown, the Executor should be stopped before dsserver is stopped.

## 3. Add a cluster

A cluster is the network address that the client sees. This address will be advertised on the network.

To configure the cluster, specify the address of the cluster (in our example `www.itso.ibm.com` or `10.20.10.100`), type:

```
dscontrol cluster add www.itso.ibm.com
```

## 4. Add a port

To add the HTTP port protocol to the Dispatcher configuration, type:

```
dscontrol port add www.itso.ibm.com:80
```

## 5. Add servers

Add each of the Web servers (web1 and web2) to the Dispatcher configuration, by typing:

```
dscontrol server add www.itso.ibm.com:80:web1  
dscontrol server add www.itso.ibm.com:80:web2
```

## 6. Configure the server to accept traffic

To configure the server to accept traffic for the cluster address, type:

```
dscontrol cluster configure www.itso.ibm.com
```

7. Start the manager

To start the manager function of Dispatcher, type:

```
dscontrol manager start
```

The manager's role is to dynamically set the “weights” the Executor uses to perform load balancing. When running by itself (without the manager), the Executor performs what is called *weighted round robin* load balancing, based upon static weights that can be manually set by the user. When starting the manager, it adjusts the weights in a dynamic fashion, based upon several statistical inputs. This provides a much more dynamic load balancing process.

8. Add the advisor

Start the advisor function of Dispatcher, to make sure that client HTTP requests to port 80 are not sent to a failed Web server, by typing:

```
dscontrol server start http 80
```

The role of the advisor component is to check the health of each server by performing a client request, and then inform the manager of the results, which uses it as input to set weights in the Executor. When an advisor detects that a server has died, the Executor will not forward any new requests to it until it is up again.

### 4.3.5 Testing the configuration

Dispatcher provides reports that can be used to verify the configuration. You can see whether the back-end servers that make up the cluster are active and sending responses to the advisors. You can also see if the traffic is being balanced using the server monitor on the GUI.

Example 4-1 shows the output of the **dscontrol manager report** command. The first table lists the back-end servers being load balanced and their status. The second table lists the servers by port, weight, number of active and new connections, and load values.

The last table shows the advisors that were started, the port, and the timeout value attributed to it.

*Example 4-1 The dscontrol manager report*

```
#dscontrol manager report
5630-A36, (C) COPYRIGHT International Business Machines Corp., 2001, 2002
All Rights Reserved * Licensed Materials - Property of IBM
```

SERVER	IP ADDRESS	STATUS
--------	------------	--------

	web2	10.20.10.104	ACTIVE
	web1	10.20.10.103	ACTIVE

MANAGER REPORT LEGEND	
ACTV	Active Connections
NEWC	New Connections
SYS	System Metric
NOW	Current Weight
NEW	New Weight
WT	Weight
CONN	Connections

www.itso.ibm.com						
10.20.10.100	WEIGHT	ACTV	NEWC	PORT	SYS	
PORT: 80	NOW NEW	49%	50%	1%	0%	
web2	9 9	0	0	190	0	
web1	10 10	0	0	10	0	

ADVISOR	CLUSTER:PORT	TIMEOUT
http	80	unlimited

You can check whether packets are being forwarded to the cluster by issuing the **dscontrol executor report** command, which produces a report of the packet traffic on the Executor component of Dispatcher, as shown in Example 4-2.

*Example 4-2 The dscontrol Executor report*

```
#dscontrol executor report
5630-A36, (C) COPYRIGHT International Business Machines Corp., 2001, 2002
All Rights Reserved * Licensed Materials - Property of IBM

Executor Report:
-----
Version level ..... 05.00.00.00 - 20021111-014835
Total packets received since starting ..... 1,202,275
Packets sent to nonforwarding address ..... 0
Packets processed locally on this machine ..... 0
Packets sent to collocated server ..... 0
Packets forwarded to any cluster ..... 1,066,105
Packets not addressed to active cluster/port .. 133,129
KBytes transferred per second ..... 0
```

```

Connections per second ..... 0
Packets discarded - headers too short ..... 0
Packets discarded - no port or servers ..... 0
Packets discarded - network adapter failure ... 0
Packets with forwarding errors..... 0

```

---

We can use the server monitor on the GUI to graphically view the load being distributed among the servers:

1. Right-click the port to be monitored and, in our case, select **Port: 80 -> Monitor....**
2. The server monitor chart window will be displayed, as shown in Figure 4-8.

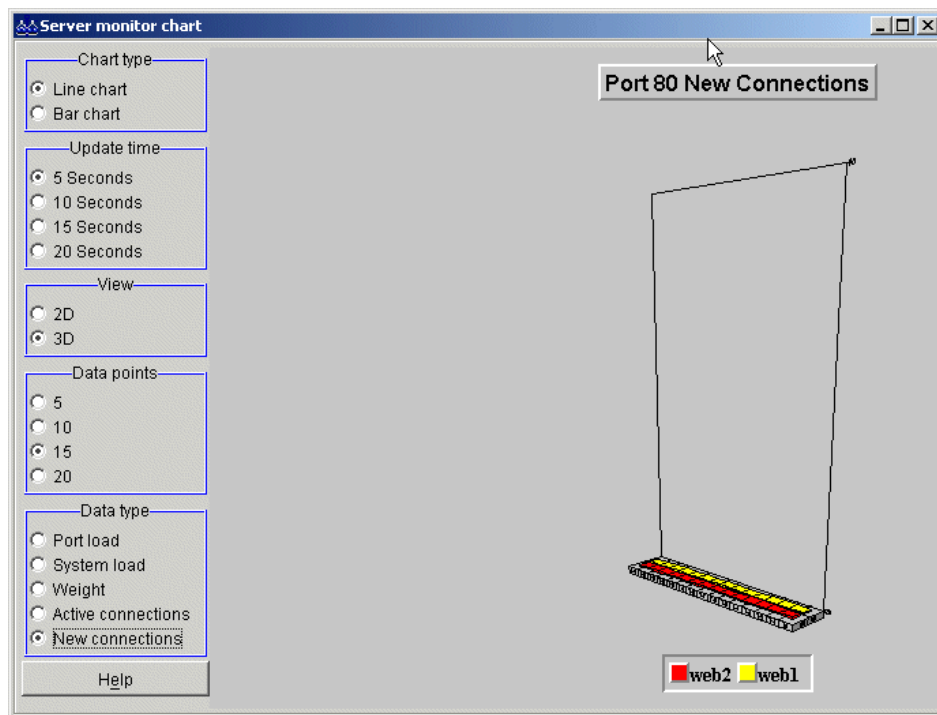


Figure 4-8 Server Monitor chart

The line chart represents the new connections value for each server on the cluster, in idle state.

To monitor the behavior of the cluster, you can try repeatedly selecting a clustered page using the browser, or you can use an HTTP benchmarking tool to send requests to the cluster. In our lab, we used ApacheBench, a benchmark tool shipped with IBM HTTP Server and Apache UNIX distributions, to issue a burst of requests to the home page of our cluster:

```
/usr/HTTPServer/bin/ab -n 5000 -c 4 http://www.itso.ibm.com/
```

This means we sent a burst of 5000 requests with concurrency of four multiple requests. The chart in Figure 4-9 shows the distribution of new requests among the servers.

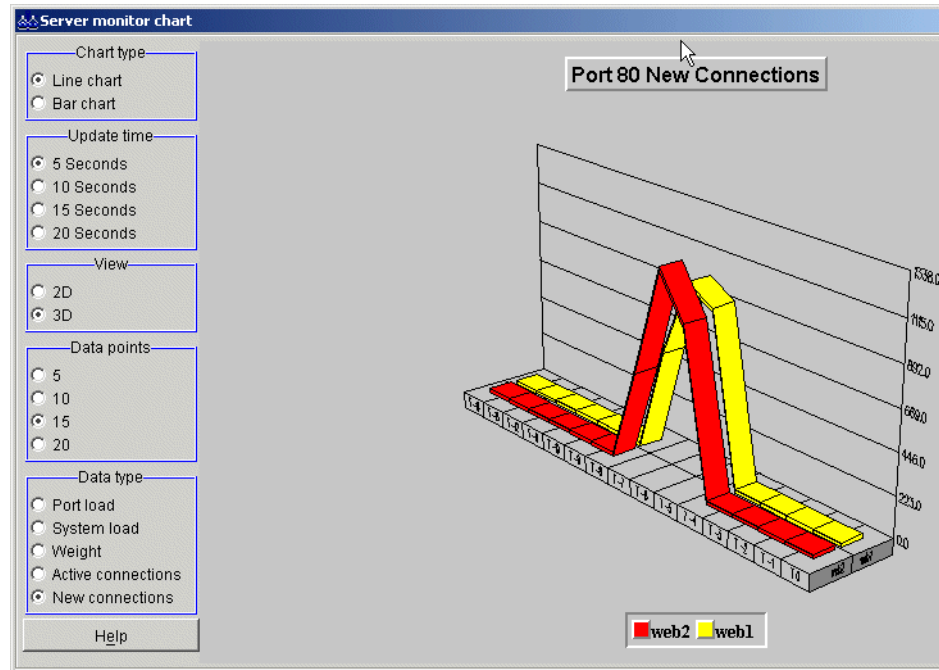


Figure 4-9 Request distribution to Web servers

## 4.4 Advisors

Advisors are lightweight clients that run inside the Dispatcher, providing information about the load of a given server. The product provides protocol-specific advisors for most popular protocols (HTTP, HTTPS, SSL, FTP, Ping, DB2, POP3, DNS, Telnet, and others).

Standard advisors send transactions periodically to determine the status of the servers (for example, for HTTP an HTTP HEAD request is sent, and for FTP, a SYST). If the transaction succeeds, the server is considered up.

To obtain the load value, the advisor:

1. Pings and opens a connection with each server.
2. Sends a protocol-specific request message.
3. Listens for a response from the server.
4. Calculates the load value.

After getting the response, the advisor makes an assessment of the server. To calculate this “load” value, most advisors measure the time for the server to respond, and then use this value (in milliseconds) as the load.

You may also set the connecttimeout and receivetimeout. The connecttimeout is the amount of time the advisor will wait before aborting the connection and the receivetimeout is the amount of time the advisor will wait before giving up on the data over the socket. The combined time in most cases is the advisor load.

5. Reports the load value to the manager function.

If the server does not respond, the advisor returns a negative value (-1) for the load.

The manager function obtains the load value reported by the advisor, which is available in the manager report, in the Port column. The manager obtains these values from all of its sources and sets proportional weight values for the Executor function.

A down server is given a weight of zero, and packets will not be forwarded to it until the server responds to the advisor again.

## Customizable advisor settings

An additional feature available since WebSphere Edge Components V5.0 is that you are now able to customize advisor settings for the HTTP and HTTPS advisors. This enables an administrator to set a request type and expected response header per server without writing any code. For example, you can request a different request such as GET /index.html and parse the response for the intended header.

An example would be:

```
dscontrol>>server set :80:was5-d advisorrequest "GET / HTTP/1.1\r\nHost:
myhost.raleigh.ibm.com\r\nConnection: Keep-Alive\r\n"
dscontrol>>server set :80:was5-c advisorresponse "200"
```

The HTTPS advisor fully negotiates an SSL connection and issues the request/response encrypted to the backend server.

Customizable advisors are an alternative to the custom advisors (discussed in “Custom advisors” on page 112) when you don't need the extensive functionality that the custom advisors give you.

### 4.4.1 Custom advisors

You can also write your own advisors for specific applications. These are called *custom advisors*, and can be based on sample Java code provided with the product. You can find the sample code in the <install\_path>/servers/samples/CustomAdvisors directory, where <install\_path> is the Load Balancer installation path (such as /opt/ibm/edge/lb on AIX, or C:\Program Files\IBM\edge\lb on Windows).

Custom advisors execute on the Dispatcher node, and must be written in the Java language and compiled with a Java compiler for the Dispatcher machine.

**Important:** For the Edge Components that are part of IBM WebSphere Application Server Network Deployment V5.1 you need a Java 1.4 compiler.

For IBM WebSphere Application Server Network Deployment V5.0, you need a Java 1.3 compiler.

Class file names must follow the form ADV\_*name*.class, where *name* is the name you choose for the advisor.

Using the Java SDK, the compile command might be:

```
javac -classpath <install_path>/servers/lib/ibmnd.jar ADV_name.java
```

**Note:** The Load Balancer base classes, found in ibmnd.jar, must be referenced in the classpath during compilation.

The advisor code must then be copied to the <install\_path>/servers/lib/CustomAdvisors directory.

To start the customer advisor, the manager function of Dispatcher must have been started. Then issue the following command to start your custom advisor:

```
dscontrol advisor start <name> <port_number>
```

The <port\_number> specified in the command is the port on which the advisor opens a connection with the target server.



More detailed information about custom advisors, describing how they work, how to write, compile and test them, including examples, development techniques, and interface methods, can be found in the *Load Balancer Administration Guide*, GC09-4602.

#### 4.4.2 Using WebSphere Application Server sample custom advisor

A sample custom advisor for WebSphere Application Server is included in the <install\_path>/servers/samples/CustomAdvisors directory.

Two files are used to implement the advisor:

- ▶ ADV\_was.java: The custom advisor code that should be compiled and executed on the Network Dispatcher machine.
- ▶ LBAAdvisor.java.servlet: The servlet code (it must be renamed to LBAAdvisor.java) that should be compiled and installed on the WebSphere Application Server machine.

The servlet returns the following string in case of success:

```
LBAAdvisor/0.92 100 Fri Dec 13 16:05:35 EST 2002
```

The advisor parses the string to get the WebSphere status (100) and informs the Dispatcher monitor.

**Notes:** We modified the Java file ADV\_was.java to reflect our settings.

- To reflect the new advisor name, the line:

```
static final String ADV_WAS_NAME = "was";
```

was changed to:

```
static final String ADV_WAS_NAME = "was5";
```

- To reflect the response of IBM HTTP Server on AIX, the line:

```
static final String HTTP_GOOD = "HTTP/1.1 200 ok";
```

was changed to:

```
static final String HTTP_GOOD = "HTTP/1.1 200 OK";
```

- To reflect the location, where you have deployed the LBAdvisor servlet, the line:

```
static final String SERVLET_LOCATION = "/servlet/LBAdvisor;
```

was changed to:

```
static final String SERVLET_LOCATION = "advisor/servlet/LBAdvisor;
```

To check whether the application server was able to serve a servlet request, we installed a simplified version of the advisor servlet on our sample topology, which always returned a status of 100, following these steps:

1. Create an application server for the advisor on server app1 (Advisor1).
2. Configure and deploy the advisor servlet to respond to `http://10.20.10.100/advisor/servlet/LBAdvisor`. The advisor used the address configured for the cluster (10.20.10.100) to request the servlet, so it had to be included as a host alias for the application server virtual host on our WebSphere configuration.
3. Create a cluster (AdvisorCluster), add existing application server Advisor1 as a cluster member, then add a new cluster member (Advisor2) on server app2.
4. Regenerate the plug-in configuration.
  - Transfer the plug-in configuration file, `<WAS_HOME>/config/cells/plugin-cfg.xml`, to the corresponding directories on the Web server machines (web1 and web2 in our case).
5. In order not to use plug-in workload management to access the advisor servlet, configure the transport information tags of the plugin-cfg.xml files on each machine, so that each Web server sends requests to one application server.

We located the original cluster definition in the plugin-cfg.xml file, as shown in Example 4-3 on page 115.

#### *Example 4-3 Original cluster definition in plugin-cfg.xml*

---

```
<ServerCluster Name="AdvisorCluster">
  <Server CloneID="u85bjrri" LoadBalanceWeight="2" Name="Advisor2">
    <Transport Hostname="app2" Port="9083" Protocol="http"/>
    <Transport Hostname="app2" Port="9446" Protocol="https">
      <Property name="keyring" value="C:\Program
Files\WebSphere\AppServer\etc\plugin-key.kdb"/>
      <Property name="stashfile" value="C:\Program
Files\WebSphere\AppServer\etc\plugin-key.sth"/>
    </Transport>
  </Server>
  <Server CloneID="u85bjt1l" LoadBalanceWeight="2" Name="Advisor1">
    <Transport Hostname="app1" Port="9085" Protocol="http"/>
    <Transport Hostname="app1" Port="9448" Protocol="https">
      <Property name="keyring" value="C:\Program
Files\WebSphere\AppServer\etc\plugin-key.kdb"/>
      <Property name="stashfile" value="C:\Program
Files\WebSphere\AppServer\etc\plugin-key.sth"/>
    </Transport>
  </Server>
  <PrimaryServers>
    <Server Name="Advisor2"/>
    <Server Name="Advisor1"/>
  </PrimaryServers>
</ServerCluster>
```

---

We directed all the HTTP requests from web1 to app1:9085 and HTTPS requests to app1:9448, as shown in Example 4-4.

#### *Example 4-4 New cluster definition on server web1*

---

```
<ServerCluster Name="AdvisorCluster">
  <Server CloneID="u85bjt1l" LoadBalanceWeight="2" Name="Advisor1">
    <Transport Hostname="app1" Port="9085" Protocol="http"/>
    <Transport Hostname="app1" Port="9448" Protocol="https">
      <Property name="keyring" value="C:\Program
Files\WebSphere\AppServer\etc\plugin-key.kdb"/>
      <Property name="stashfile" value="C:\Program
Files\WebSphere\AppServer\etc\plugin-key.sth"/>
    </Transport>
  </Server>
  <PrimaryServers>
    <Server Name="Advisor1"/>
  </PrimaryServers>
</ServerCluster>
```

---

We also directed all the HTTP requests from web2 to app2:9083 and HTTPS requests to app2:9446, as shown in Example 4-5.

*Example 4-5 New cluster definition on server web2*

---

```
<ServerCluster Name="AdvisorCluster">
  <Server CloneID="u85bjrri" LoadBalanceWeight="2" Name="Advisor2">
    <Transport Hostname="app2" Port="9083" Protocol="http"/>
    <Transport Hostname="app2" Port="9446" Protocol="https">
      <Property name="keyring" value="C:\Program
Files\WebSphere\AppServer\etc\plugin-key.kdb"/>
      <Property name="stashfile" value="C:\Program
Files\WebSphere\AppServer\etc\plugin-key.sth"/>
    </Transport>
  </Server>
  <PrimaryServers>
    <Server Name="Advisor2"/>
  </PrimaryServers>
</ServerCluster>
```

---

6. Test if the servlet is responding using  
`http://10.20.10.100/advisor/servlet/LBAdvisor?debug=1`. This option returns a HTML/text page with some diagnostic information.
7. Restart Load Balancer server:  

```
dsserver stop
dsserver start
```
8. Start the custom advisor using the command:  

```
dscontrol advisor start was5 80
```

where was5 is your custom advisor name and 80 is the port number, where the advisor opens a connection with the target server.
9. Start the GUI of Load Balancer and select your new advisor, then select the **Current Statistics** tab on the right. The advisor status is displayed. Click **Refresh** to update the status. If the advisor is working properly, the load value is displayed. If there is no response, a status of -1 is returned (see Figure 4-10 on page 117).

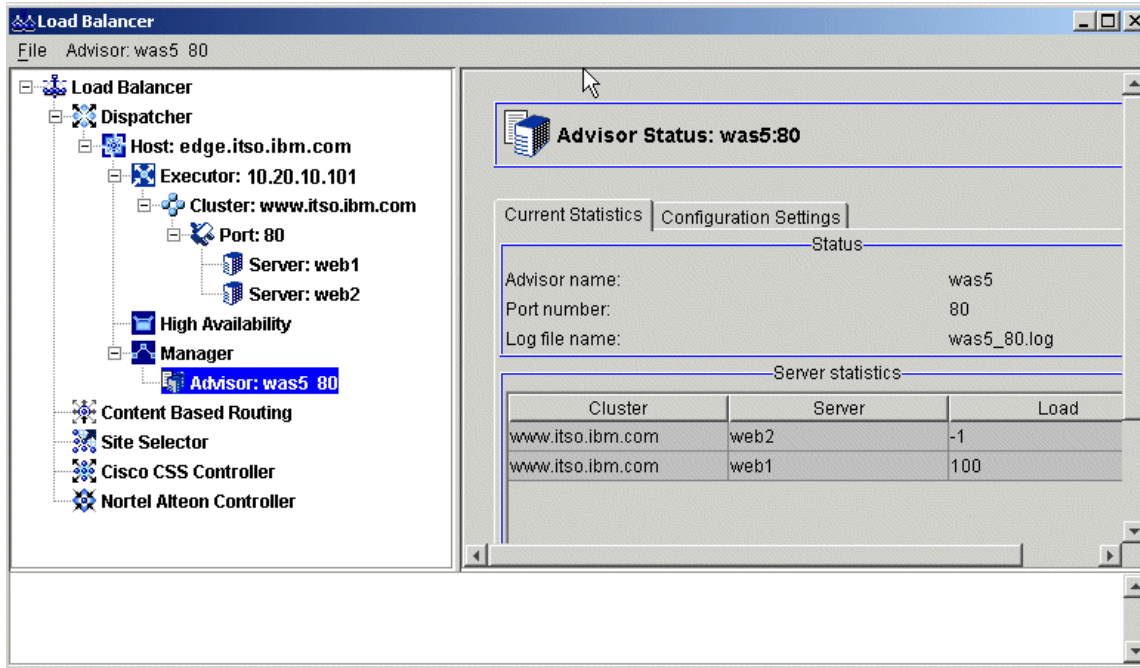


Figure 4-10 WebSphere advisor status

If there is no response, one option is to select the **Configuration Settings** pane and change the advisor logging level to Advanced, for example, then look at the log file at <install\_path>/servers/logs/dispatcher. A lot of detailed information is recorded, showing all requests and responses, which may help to locate where the problem is occurring.

If we simulate a failure by stopping the application server on host was2 or disconnecting the network cable, we can see how the requests are directed to the available application server (associated to the web1 server). We can see the graphical statistics of the traffic by selecting the **Monitor** option under the cluster the advisor is associated with, as shown in Figure 4-11 on page 118.

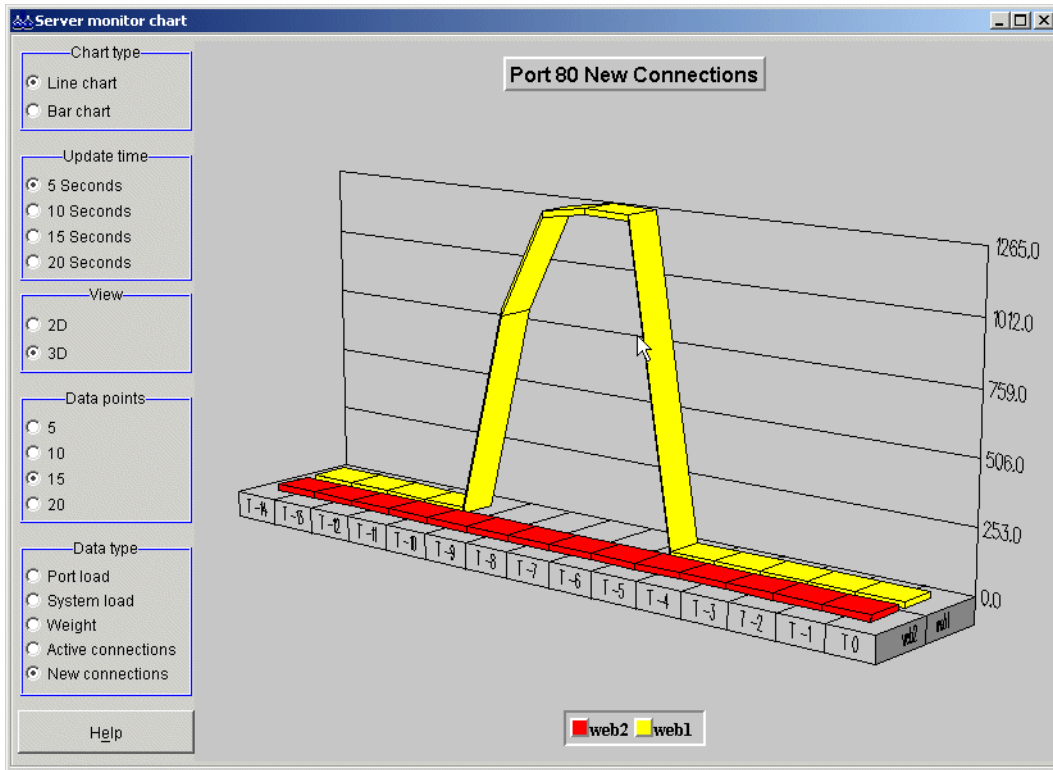


Figure 4-11 Request distribution after one application server becomes unavailable

## 4.5 Server affinity

Server affinity is a technique that enables the Load Balancer to remember which server was chosen for a certain client at his initial request. Subsequent requests are then directed to the same server again.

If the affinity feature is disabled when a new TCP/IP connection is received from a client, Load Balancer picks the right server at that moment and forwards the packet to it. If a subsequent connection comes in from the same client, Load Balancer treats it as an unrelated connection, and again picks the most appropriate server at that moment.

Server affinity allows load balancing for those applications that need to preserve state across distinct connections from a client. Maintaining state is a requirement of many application encountered on the Internet today, including “shopping carts”, home banking, and so on.

Some options available to maintain application state based on server affinity are:

- ▶ “Stickyness” to source IP address
- ▶ Passive cookie
- ▶ Active cookie
- ▶ URI
- ▶ SSL session ID

The passive cookie, active cookie, and URI affinity options are rules based. They depend on the content of the client requests.

**Important:** Not all options are available for all components. For example, “Stickyness” to source IP address is the *only* affinity available for the Dispatcher component.

We have noted at the end of each option for which components this affinity option applies.

#### 4.5.1 “Stickyness” to source IP address

This affinity feature is enabled by configuring the clustered port to be sticky. Configuring a cluster’s port to be sticky allows subsequent client requests to be directed to the same server. This is done by setting the sticky time to a positive number; the feature can be disabled by setting the sticky time to zero.

This feature applies to the Dispatcher, the CBR, and the Site Selector components of Load Balancer.

**Note:** This affinity strategy has some drawbacks. Some ISPs use proxies that collapse many client connections into a small number of source IP addresses. A large number of users who are not part of the session will be connected to the same server. Other proxies use a pool of user IP addresses chosen at random, even for connections from the same user, invalidating the affinity.

#### 4.5.2 Passive cookie affinity

Passive cookie affinity is based on the content of cookies (name/value) generated by the HTTP server or by the application server. You must specify a cookie name to be monitored by Load Balancer in order to distinguish which server the request is to be sent to.

If the cookie value in the client request is not found or does not match any of the servers’ cookie values, the server will be chosen using the weighted round robin technique.

This feature applies to both the CBR component and to the Dispatcher component's *cbr* forwarding method.

### 4.5.3 Active cookie affinity

Active cookie affinity enables load balancing Web traffic with affinity to the same server based on the cookies generated by the Load Balancer. This function is enabled by setting the sticky time of a rule to a positive number, and setting the affinity to cookie. The generated cookie contains:

- ▶ The cluster, port, and rule
- ▶ The server that was load balanced to
- ▶ A timeout time stamp for when the affinity is no longer valid

The active cookie affinity feature applies only to the CBR component.

### 4.5.4 URI

URI affinity allows you to load balance Web traffic to caching proxy servers, which allow unique content to be cached on each individual server. As a result, you will effectively increase the capacity of your site's cache by eliminating redundant caching of content on multiple machines. You can configure URI affinity at the rule level and once it is enabled and the servers are running, then the Load Balancer will forward new incoming requests with the same URI to the same server.

URI affinity applies to the CBR component and to Dispatcher component's *cbr* forwarding method.

### 4.5.5 SSL session ID

During establishment of an SSL encrypted session, a handshake protocol is used to negotiate a session ID. This handshaking phase consumes a good deal of CPU power, so directing HTTPS requests to the same server, using the already established SSL session, saves processing time and increases the overall performance of the HTTP server.

Load Balancer watches the packets during the handshake phase and holds information about the session ID if SSL session negotiation is detected.

The forwarding method used to configure SSL session ID affinity is the kernel-based content-based routing function called *cbr forwarding*.



The steps for configuring this are as follows:

1. First configure the client gateway address of the Executor as shown in Figure 4-12.

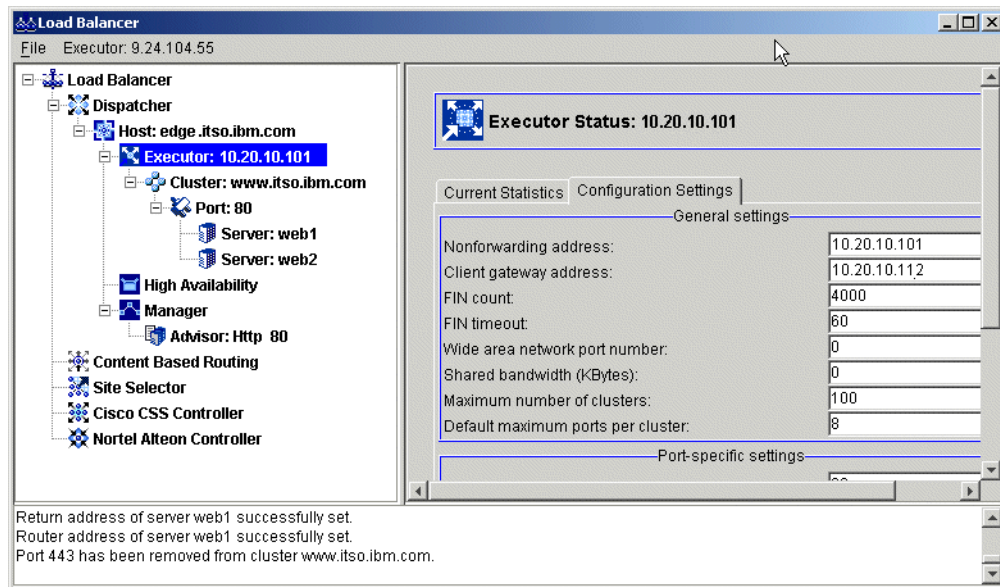


Figure 4-12 Client gateway address configuration

2. Add port 443, using content-based routing as the forwarding method, as shown in Figure 4-13 on page 122.

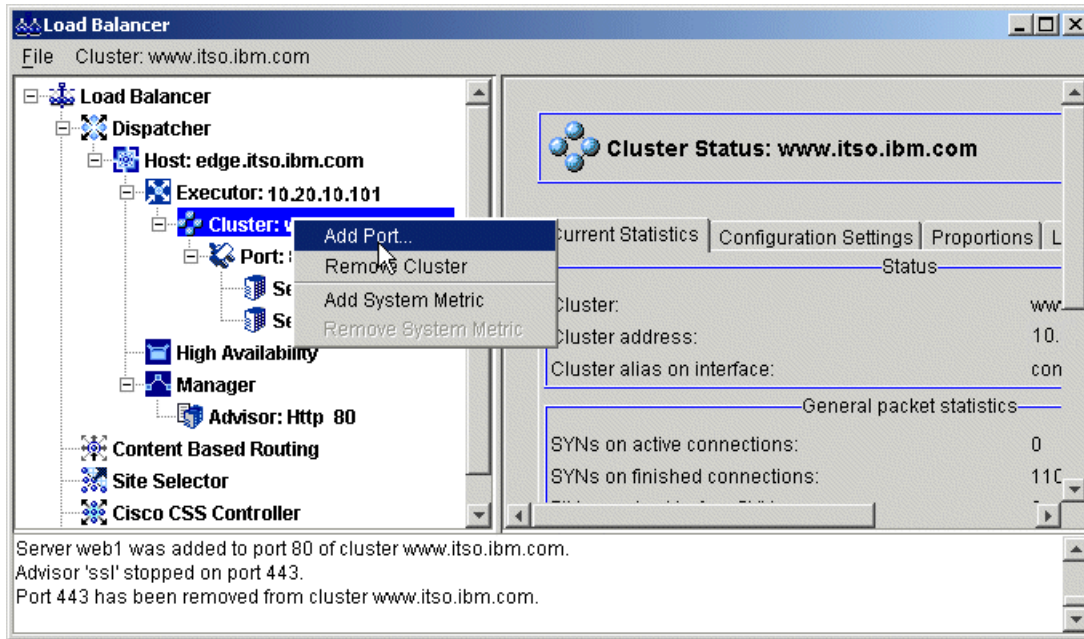


Figure 4-13 Adding a port

3. Add the servers to be load balanced on port 443, as illustrated in Figure 4-14.

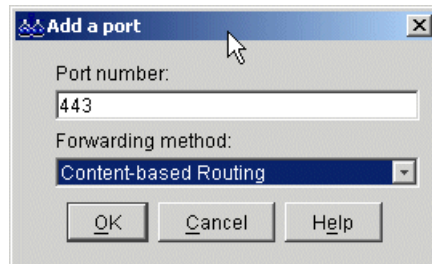


Figure 4-14 Setting the port number and forwarding method

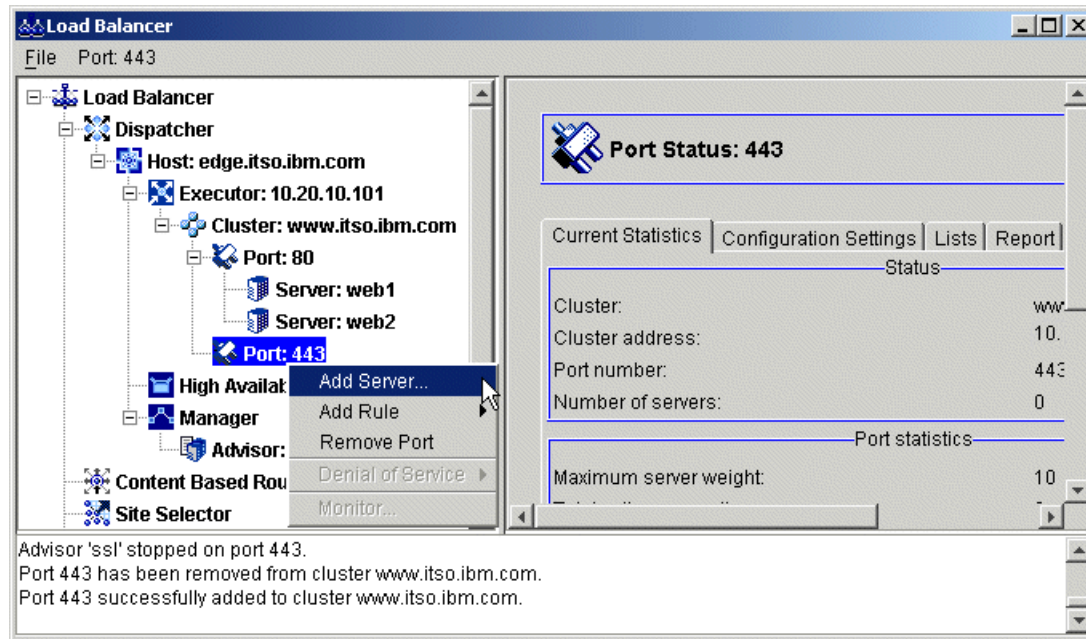


Figure 4-15 Adding a server to the port

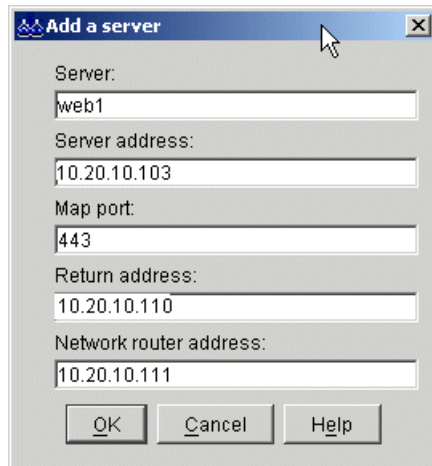
4. Add the server address, return address, and network router address.

The return address is a unique address or host name that is configured on the Dispatcher machine. Dispatcher uses the return address as its source address when load balancing the client's request to the server. This ensures that the server will return the packet to the Dispatcher machine rather than sending the packet directly to the client. The return address cannot be the same as the cluster address, server address, or non-forwarding address (NFA). We added a new alias to the network interface (10.20.10.110) to be configured as the return address.

AIX: `ifconfig en0 alias 10.20.10.110 netmask 255.255.0.0`

Windows: `dsconfig en0 alias 10.20.10.110 netmask 255.255.0.0`

The network router address is the address for the router to access the remote server. If this is a locally attached server, enter the server address. We set the router address to our default gateway address 10.20.10.111, as shown in Figure 4-16 on page 124.



*Figure 4-16 Configuration and addresses for cbr forwarding*

5. Configure an SSL advisor to query for servers listening on port 443, by selecting the Manager **Start Advisor** option (start Manager if necessary) as shown in Figure 4-17 on page 125.

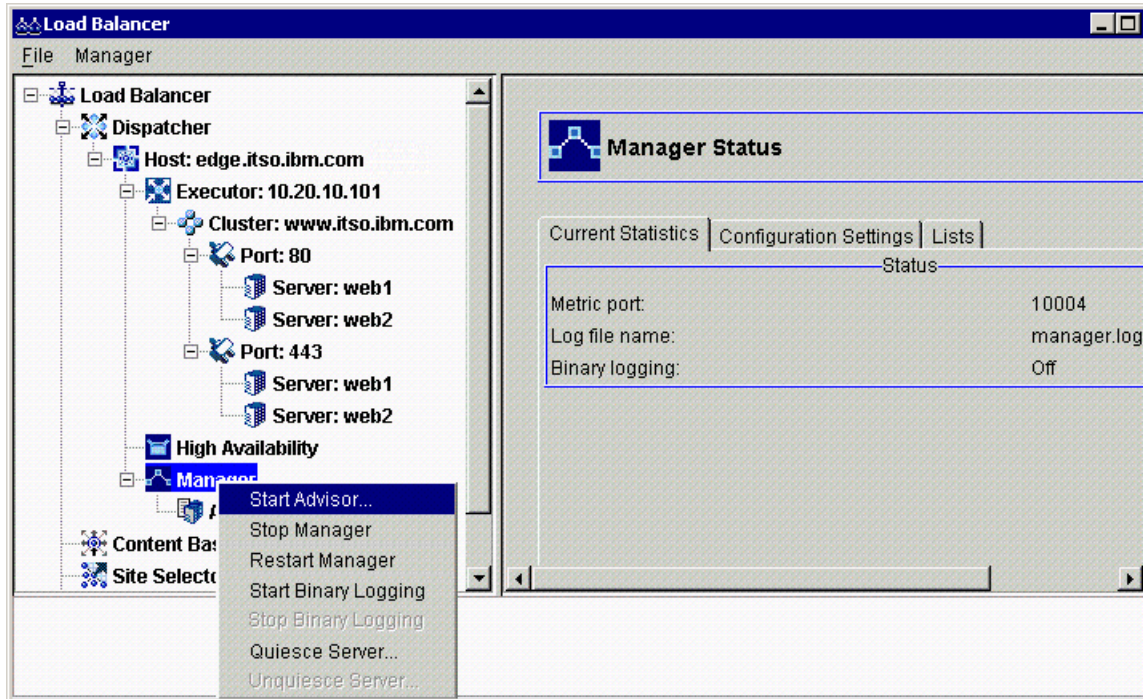


Figure 4-17 Starting an advisor

6. Select **SSL** as the advisor name, as illustrated in Figure 4-18.

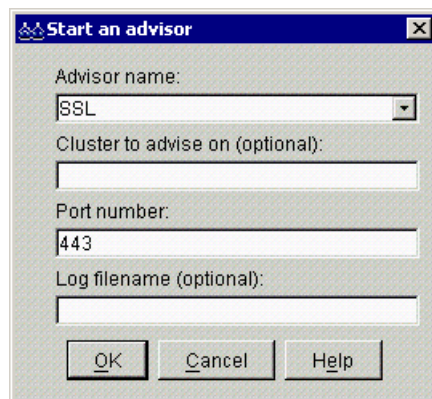


Figure 4-18 Configuring SSL advisor



7. Set up SSL session ID affinity.

To set up SSL session ID affinity, you need to configure a sticky time value, such as 60 seconds, to port 443, as illustrated in Figure 4-19.

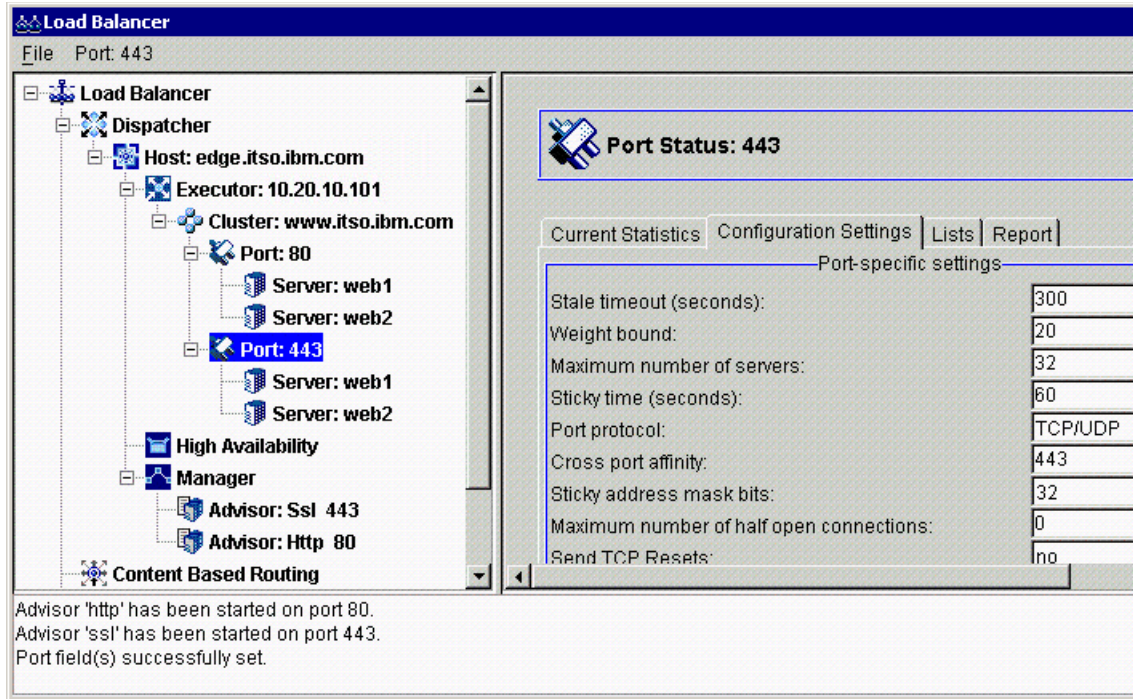


Figure 4-19 Configuring sticky time

The graph in Figure 4-20 on page 127 shows the traffic directed to the cluster when SSL session ID affinity for port 443 is configured. All connections are dispatched to the same server.

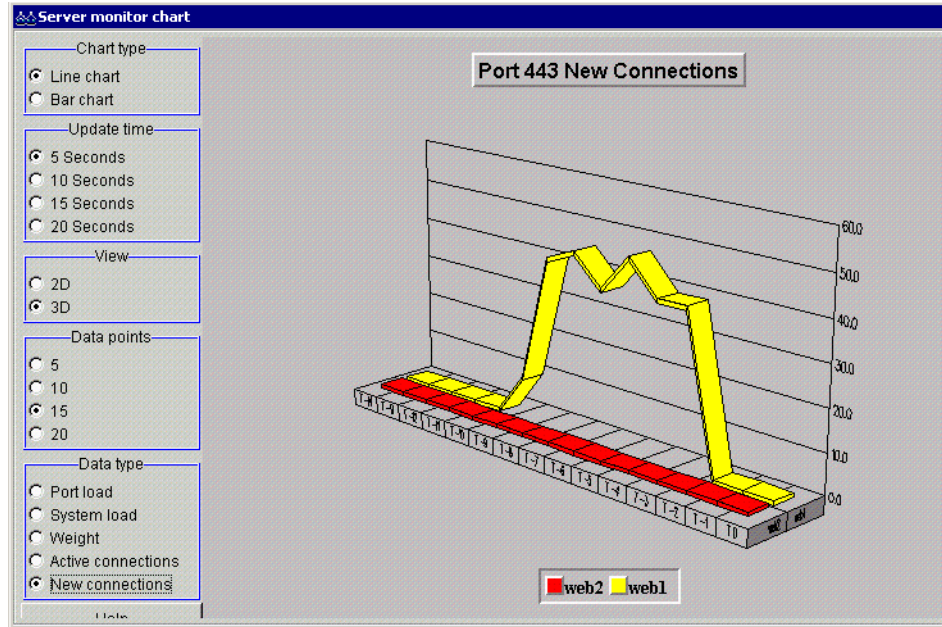


Figure 4-20 Request distribution using SSL session ID affinity

## 4.6 Caching Proxy

Caching Proxy intercepts requests from the client, retrieves the requested information from the content-hosting machines, and delivers that information back to the client. You can configure Caching Proxy to handle such protocols as HTTP, FTP, and Gopher.

The Caching Proxy stores cachable content in a local cache before delivering it to the requestor. Examples of cachable content include static Web pages and dynamic Web pages with infrequently changing fragments. Caching enables the Caching Proxy to satisfy subsequent requests for the same content by delivering it directly from local cache, which is much quicker than retrieving it again from the content host.

The Caching Proxy can be configured as Reverse, Forward and Transparent Proxy Server. The cache can be stored on physical storage devices or in memory.

Caching Proxy can be administered through a browser-based GUI or manually. You can change the settings by editing the file `ibmproxy.conf` located in the `<install_directory>\IBM\edge\cp\etc\en_US`. An administrator user ID and

password is needed to administer the Caching Proxy through a browser and for using the configuration wizard.

Use the following commands to add an administrator entry.

On UNIX:

```
# htadm -adduser /opt/ibm/edge/cp/server_root/protect/webadmin.passwd
```

When prompted, provide a user name, password and name for the administrator.

On Windows:

```
cd \Program Files\IBM\edge\cp\server_root\protect\  
htadm -adduser webadmin.passwd
```

When prompted, provide a user name, password and name for the administrator.

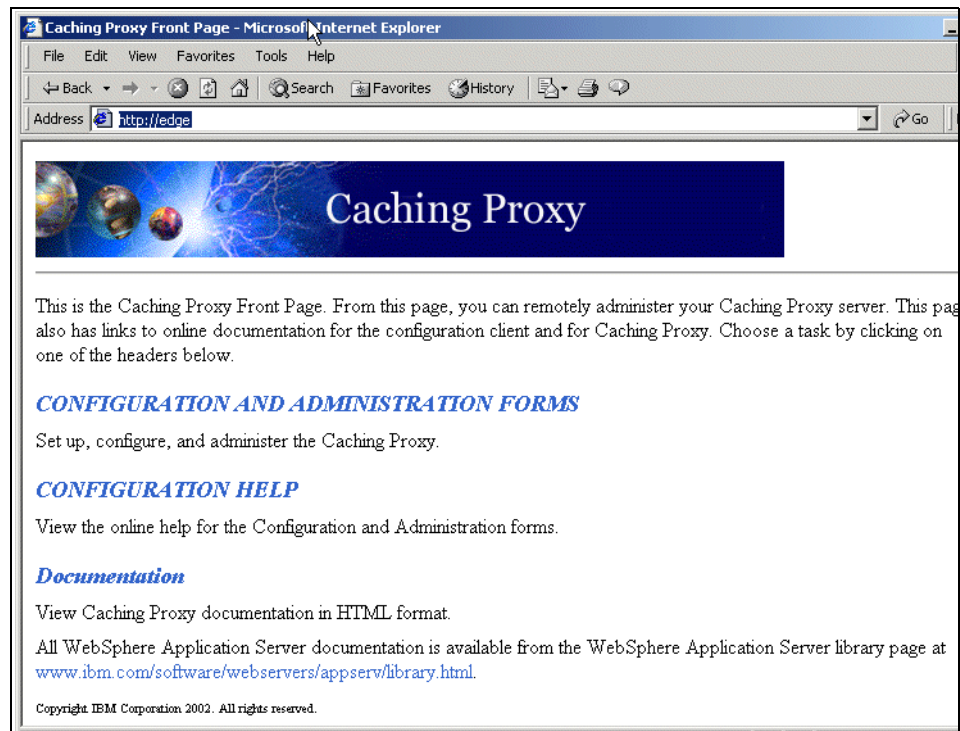


Figure 4-21 Caching Proxy Administrative Console



### 4.6.1 Forward proxy

The Caching Proxy can be configured as a *forward proxy*. Normally Internet access providers configure in this mode. When configured in this mode, it handles requests from multiple client browsers, retrieves data from the Internet and caches the retrieved data for future use. In this case, the client browser has to be configured to use the proxy server. When a client requests a page, the caching proxy forwards it to the content host located across the Internet, then caches the retrieved data and delivers the retrieved data to the client. If someone else requests the same page the next time, it will be served from the cache. This decreases the network usage and gives better response times.

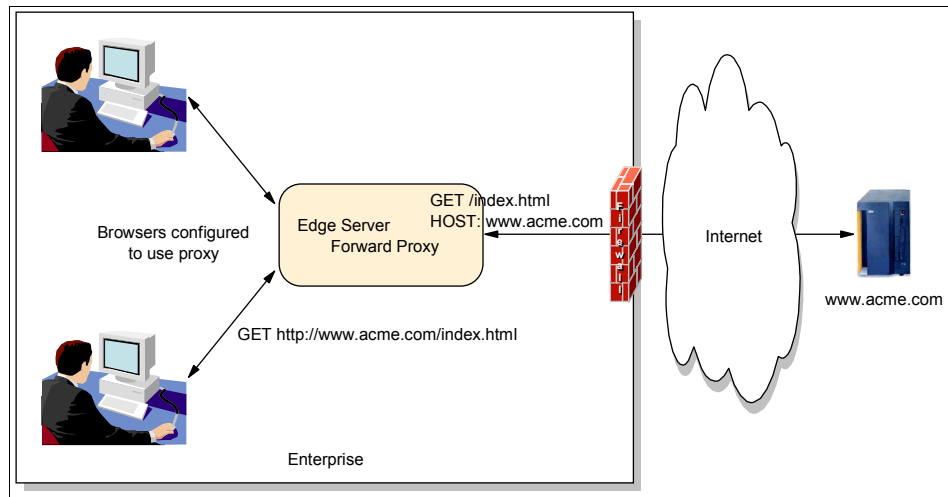


Figure 4-22 Forward proxy scenario

### 4.6.2 Reverse proxy (IP forwarding)

IP-forwarding topologies use a *reverse proxy* server, such as the Caching Proxy, to receive incoming HTTP requests and forward them to a Web server. The Web server forwards the requests to the application servers for actual processing. The reverse proxy returns completed requests to the client, hiding the originating Web server.

If a client then requests the same data next time, it will not be sent to the back-end server for processing, but instead will be served from the cache. This prevents unnecessary back-end processing for the same data requests, thus providing better response times.

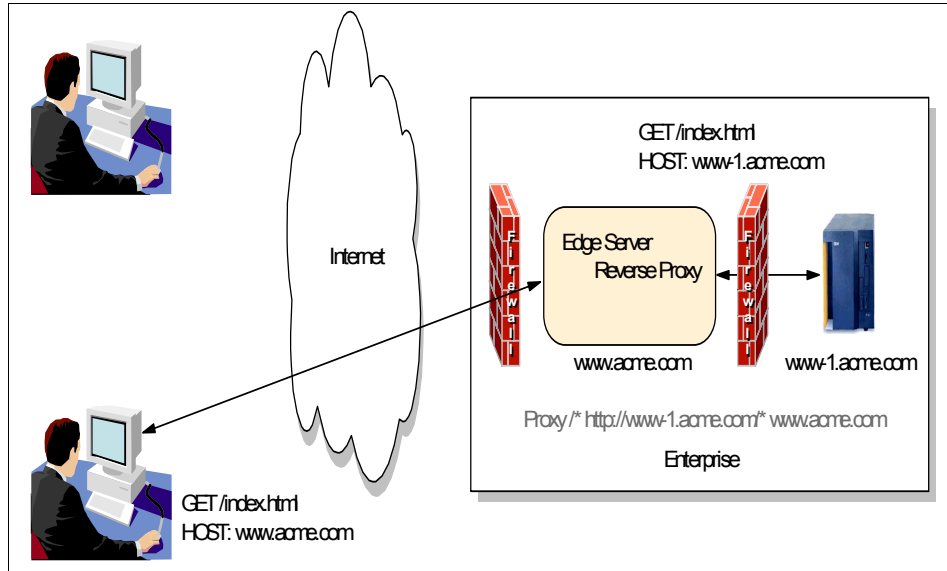


Figure 4-23 Reverse proxy scenario

### 4.6.3 Load Balancing

Multiple caching proxy servers can be configured to increase your site performance, compared with a single caching proxy at peak load times. Load Balancer's Dispatcher component can be used to distribute the load to the caching proxy servers.

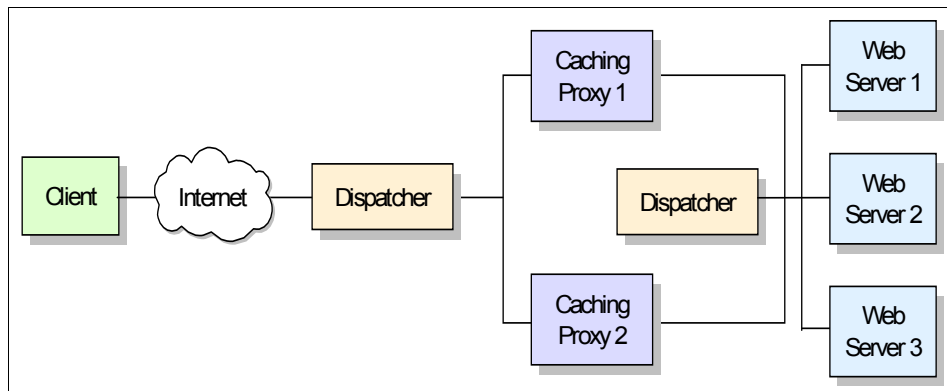


Figure 4-24 Load balancing multiple caching proxy servers

#### 4.6.4 Dynamic caching

The dynamic caching function enables the Caching Proxy to cache dynamically generated content in the form of responses from JSPs and servlets generated by an IBM WebSphere Application Server. A caching proxy adapter module is used at the application server to modify the responses, so that they can be cached at the proxy server in addition to being cached in the application server's dynamic cache. With this feature, dynamically generated content can be cached at the entry point of the network, avoiding repeated requests to the application server, when the same content is requested by multiple clients.

See Chapter 14, "Dynamic caching" on page 527 for more information about this topic and for details on how to configure the Caching Proxy to cache dynamic content.





## Plug-in workload management and failover

In this chapter, we cover how to set up the Web container for workload management. We discuss the components that make this possible, the configuration settings, and the resulting behaviors. We also cover session management in a workload-managed environment and the process of troubleshooting all these areas.

This chapter covers in detail:

- ▶ The embedded HTTP transport
- ▶ Setting up the Web containers
- ▶ WebSphere plug-in workload management
- ▶ Session management
- ▶ Troubleshooting the Web server plug-in
- ▶ Web server plug-in behavior and failover

## 5.1 Introduction

The Web container manages the J2EE components, servlets and JSPs that are accessed from the Web.

Traffic for these components is workload managed by configuring clusters within a cell in WebSphere Application Server and then configuring Web servers to access cluster members.

This workload management process also sets up failover and backup servers in the event that one or more of the cluster members should fail or all primary servers fail.

WebSphere Application Server V5.1 supports a variety of Web servers. These Web servers are necessary for directing traffic from users' browsers to the applications running in WebSphere. This is achieved by installing the WebSphere plug-in on the Web servers. As shown in Figure 5-1, the plug-in is a module that runs as part of the Web server process. It acts as a router, deciding what traffic is for WebSphere and what traffic should be handled by the Web server itself.

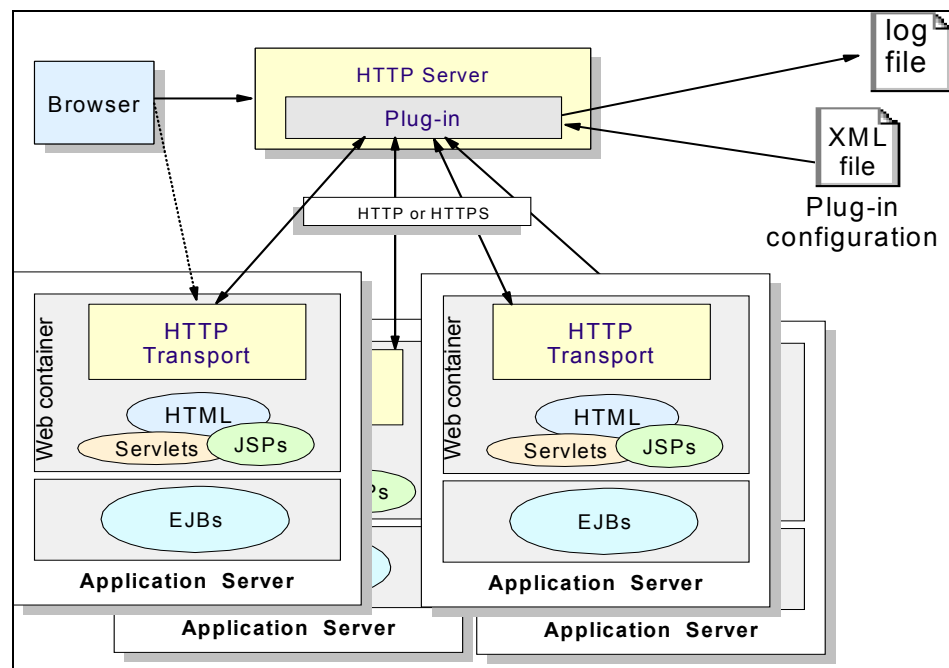


Figure 5-1 Plug-in components and interactions

The Web server plug-in uses HTTP and HTTPS as its transport protocol between the Web server and WebSphere Application Server. This has been carried over from WebSphere V4.0. However, the WebSphere Application Server V5 plug-in does add some new features, such as:

- ▶ Weighted round robin workload management
- ▶ Backup servers

In this chapter we will be using a subset of the sample topology described in Figure 7-1 on page 258 to demonstrate plug-in work load management. This subset is illustrated in Figure 5-2.

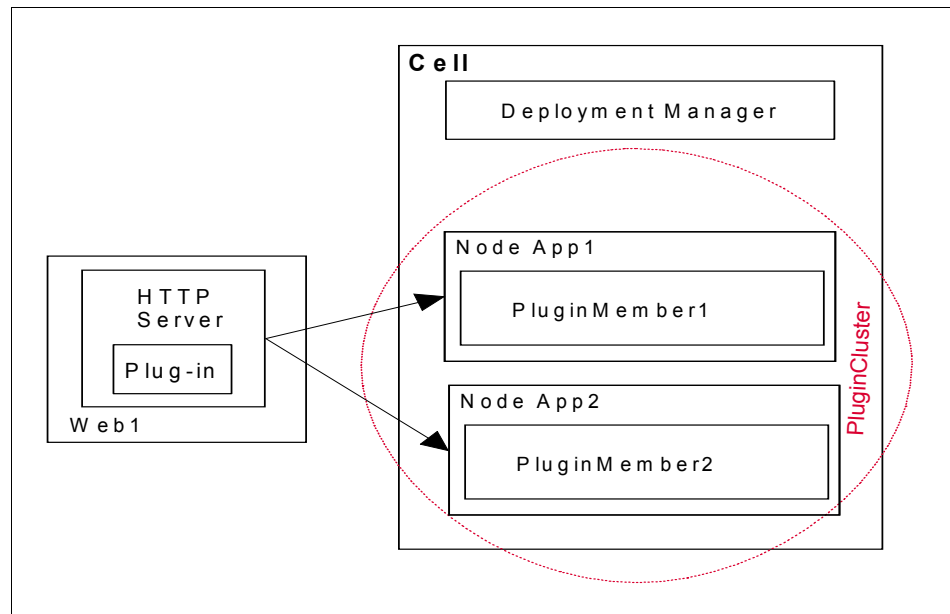


Figure 5-2 Topology used in this chapter

## 5.2 The embedded HTTP transport

Before moving on to the main part of this chapter, it is important to mention that WebSphere Application Server V5 has a built-in HTTP server, which we refer to as the embedded HTTP transport. It allows for direct connection from the user's browser to the application without the need for a separate Web server using the internal HTTP transport port. The HTTP service runs using the transport settings specified in the Web container setup, such as port 9080. You can find or

manipulate these settings in the Administrative Console by selecting **Servers -> Application Servers -> <AppServer Name> -> Web Container -> HTTP Transports**.

Using the internal transport in the Web container is useful within development environments because there is no need to configure a separate Web server or to update the plug-in when changes are made to the URL mappings.

However, it is strongly recommended that this internal Web server not be used in production environments. If the user connects directly to the embedded HTTP transport, then they bypass the plug-in and the workload management it performs, as well as the failover and session affinity mechanisms. Refer to 6.1.1, “Using the embedded HTTP transport vs. a stand-alone Web server” of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195 for more details about this.

## 5.3 Setting up the Web containers

The workload management performed by the plug-in distributes the load between Web containers. Thus, the Web containers need to be set up correctly so that the plug-in can route traffic to them.

This section discusses setting up virtual hosts, HTTP transports, clusters, and cluster members. By understanding these you will learn where and over what connection the plug-in is routing requests.

### 5.3.1 Virtual hosts

The Web containers handle traffic for the Web modules contained within them. WebSphere uses virtual hosts as filters for the traffic coming in from the Web. A virtual host is a configuration enabling a single host machine to resemble multiple host machines. Resources associated with one virtual host cannot share data with resources associated with another virtual host, even if the virtual hosts share the same physical machine.

Each virtual host has a logical name and a list of one or more DNS aliases by which it is known. A DNS alias is the TCP/IP host name and port number used to request the servlet, for example `yourHostName:80`. When no port number is specified, 80 is assumed as the default.

When a servlet request is made, the server name and port number entered into the browser are compared to a list of all known aliases in an effort to locate the correct virtual host and serve the servlet. If no match is found, an error is returned to the browser.



A virtual host is not associated with a particular node (machine). It is a configuration, rather than a “live object,” explaining why it can be created, but not started or stopped.

For many users, virtual host creation is unnecessary because, by default, WebSphere Application Server V5 provides two virtual hosts:

- ▶ The default\_host, which is used for accessing most applications
- ▶ The admin\_host, which is configured for accessing the Administrative Console (while other applications are not accessible through this virtual host)

The default settings for default\_host map to all requests for any alias on ports 80, 9443, and 9080. The default settings for admin\_host map to requests on ports 9090 and 9043.

The virtual host definition is a list of hosts and ports on which a Web module will accept traffic. As shown in Figure 5-3, the PluginVirtualHost virtual host uses a wildcard (\*) to allow traffic from all hosts on 9088, and hosts edge, web1, and web2 on port 80. The settings for virtual host aliases can be found in the Administrative Console by clicking **Environment -> Virtual Hosts -> <Virtual\_Host\_Name> -> Host Aliases**.

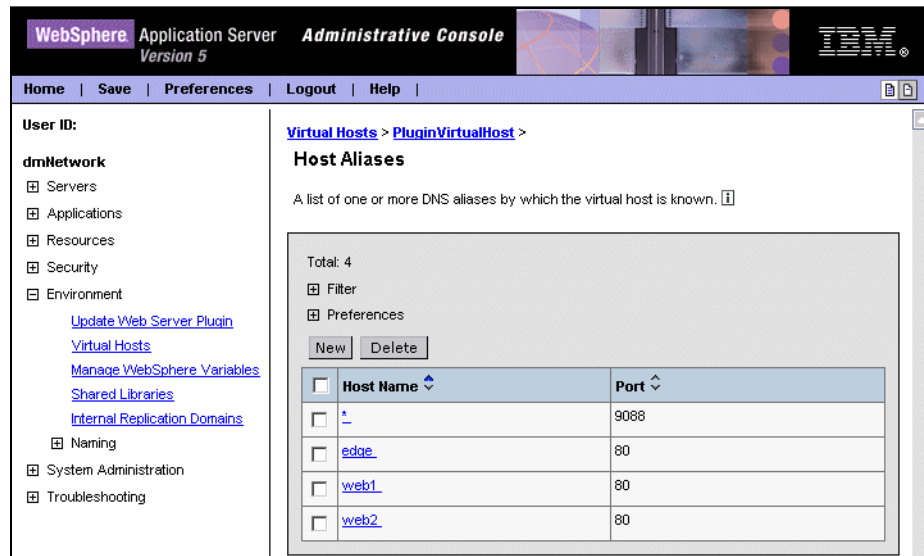


Figure 5-3 The virtual host setup

It is recommended that you specify exactly the hosts and ports for your Web container, removing wildcard settings. Removing the wildcards will prevent any Web server redirecting traffic to your Web containers.

Virtual hosts are especially useful when setting up multiple application servers that use the same URI. Virtual host filtering can be used so that the request `www.vhost1.com/snoop` and `www.vhost2.com/snoop` can be handled by two different Web modules.

For in-depth information about configuring virtual hosts, see Chapter 12, “Configuring the Environment” of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195.

**Note:** When setting up extra virtual hosts, it is important to make sure that each contains a completely unique host name and port number entries. If there are wildcards or the same host name is listed in more than one virtual host, then the application server will not be able to resolve requests and an error will occur that can be viewed in the WebSphere Administrative Console.

The error that will be seen is:

```
PLGN0021E: Servlet Request Processor Exception: Virtual Host/WebGroup Not Found : The web group /xx has not been defined
```

This is a generic error message and it can mean a number of things, but if you are positive the server is running, then check the virtual hosts.

### 5.3.2 Transports

Each application server has a Web container and transports defined for each Web container. This can be found by clicking **Servers -> Application Servers -> <AppServer Name> -> Web Container -> HTTP Transports** as shown in Figure 5-4 on page 139. As can be seen, we have two transports configured for the Web container, 9088 and 9447, where port 9447 is being used for HTTPS communication.

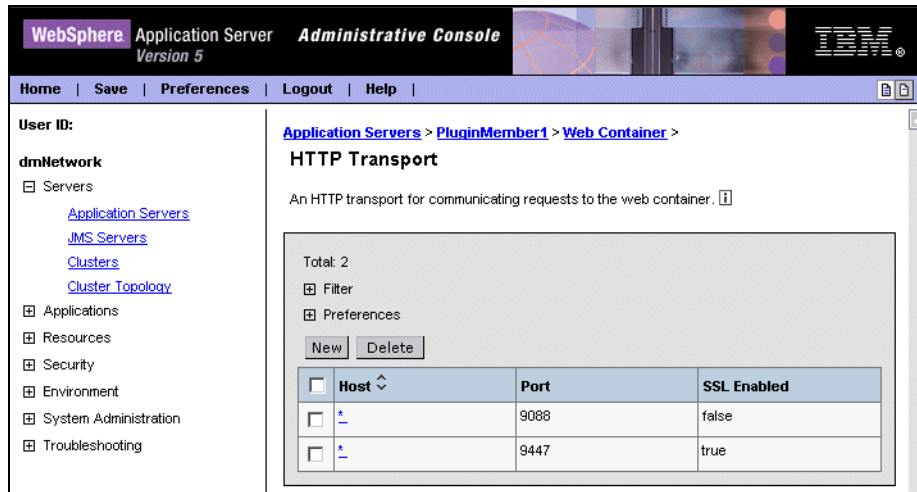


Figure 5-4 Transport settings

A “transport” defines a communication mechanism that will be used by the plug-in to access a Web container. In WebSphere Application Server V5, the protocol used for this is HTTP (or HTTPS) which runs over TCP/IP. The embedded HTTP transport within each Web container will listen on the port specified and manage traffic that is forwarded from the Web server plug-in.

It is important to understand the transports, since these settings directly affect how the plug-in works. Also, by understanding and setting up transports as the first part of a cluster member setup, time can be saved. This is because, in a workload managed environment, transport settings are unique to each node. When a cluster member is added, you have the option to generate unique port numbers for the cluster member as shown in Figure 5-5 on page 140. WebSphere Application Server will allocate it unique port numbers and define the necessary transport settings.

**WebSphere** Application Server **Administrative Console**  
Version 5

Home | Save | Preferences | Logout | Help

User ID:

**dmNetwork**

- Servers
  - Application Servers
  - JMS Servers
  - Clusters
  - Cluster Topology
- Applications
- Resources
- Security
- Environment
- System Administration
- Troubleshooting

**Create New Cluster Members**

Create New Cluster Members

→ **Step 1: Enter Basic Cluster Members Information**

Select the node where the new application server will reside.

Member name:

Select node:

Weight:

Http Ports: ☒ Generate Unique Http Ports

<input type="checkbox"/>	Application Servers	Nodes	Weight
--------------------------	---------------------	-------	--------

Step 2 Summary

Figure 5-5 Generating unique port numbers

## Setting up transports

An application server will need at most two transports defined: HTTP and/or HTTPS.

When defining a transport, there is an option to define the host name, as shown in Figure 5-6 on page 141. This setting can be reached in the Administrative Console by clicking **Servers -> Application Servers -> <AppServer Name> -> Web Container -> HTTP Transports -> <Transport\_Name>**.

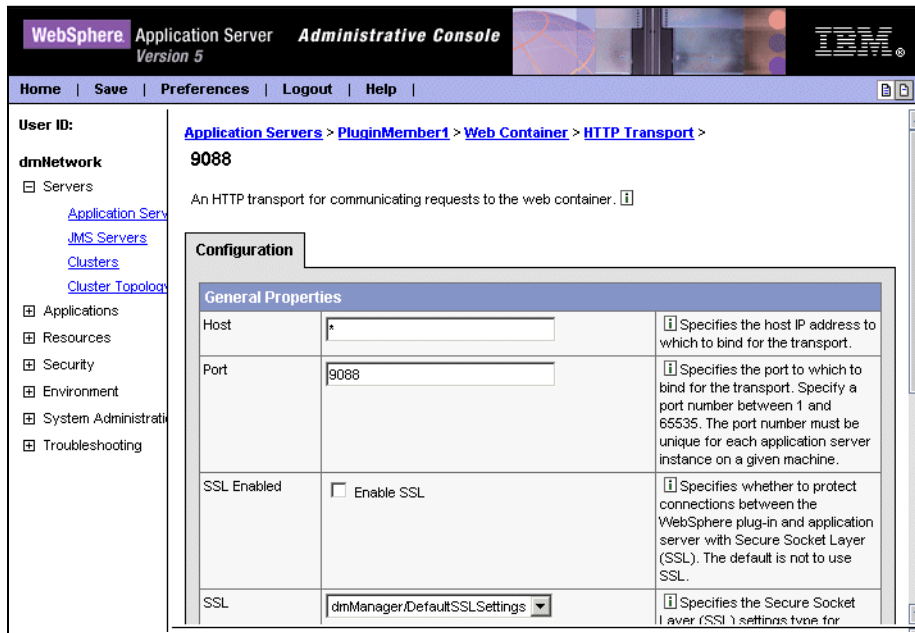


Figure 5-6 Transport properties

This is the host IP address to which the plug-in will attempt to connect. If \* is used for this, then WebSphere Application Server will replace this with the application server host name when the plug-in configuration is regenerated (see 5.4.3, “Generation of the plug-in configuration file” on page 151). This should be the method used when setting up transports in a multi-cluster member, multi-machine environment.

For a more detailed look at SSL configuration, refer to Chapter 10, “Administering WebSphere Security”, of the redbook *IBM WebSphere V5.0 Security*, SG24-6573.

## Setting up multiple transports

If you define multiple transports of the same type (for example, HTTP) for one application server, the plug-in will try to cross reference a port number specified in the VirtualHostGroup to a transport specified for a plug-in member.

An example of the type of plug-in configuration needed for this to occur is shown in Figure 5-1 on page 142.

```
.
<VirtualHostGroup Name="PluginVirtualHost">
  <VirtualHost Name="*:9088"/>
  <VirtualHost Name="web2:80"/>
</VirtualHostGroup>
  <ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="PluginCluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
    <Server CloneID="v544d031" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="8" MaxConnections="-1" Name="was1node_PluginMember1"
WaitForContinue="false">
      <Transport Hostname="app1.itso.ibm.com" Port="9088" Protocol="http"/>
      <Transport Hostname="app1.itso.ibm.com" Port="9095" Protocol="http"/>
      <Transport Hostname="app1.itso.ibm.com" Port="9099" Protocol="http"/>
    </Server>
    <Server CloneID="v544d0o0" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="was2Node_PluginMember2"
WaitForContinue="false">
      <Transport Hostname="app2.itso.ibm.com" Port="9086" Protocol="http"/>
    </Server>
  <PrimaryServers>
    <Server Name="was1node_PluginMember1"/>
    <Server Name="was2Node_PluginMember2"/>
  </PrimaryServers>
</ServerCluster>
  <UriGroup Name="PluginVirtualHost_PluginCluster_URIs">
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionId"
Name="/snoop/*"/>
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionId"
Name="/hello"/>
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionId"
Name="/hitcount"/>
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionId"
Name="*.jsp"/>
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionId"
Name="*.jsv"/>
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionId"
Name="*.jsw"/>
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionId"
Name="/j_security_check"/>
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionId"
Name="/ibm_security_logout"/>
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionId"
Name="/servlet/*"/>
  </UriGroup>
```

```
<Route ServerCluster="PluginCluster"
UriGroup="PluginVirtualHost_PluginCluster_URIs"
VirtualHostGroup="PluginVirtualHost"/>
.
```

---

Using Example 5-1 on page 142, if a request to `http://web2/snoop` was sent to the plug-in and `PluginMember1` was chosen for the request, the request would be sent to `app1.itso.ibm.com` on port 9088. This is because the plug-in has cross referenced the ports specified in the transports of *PluginMember1* and the ports specified in the *VirtualHostGroup PluginVirtualHost*. The match between the two is port 9088; hence the request is sent to the transport on port 9088. If no match is found, then the last transport for `PluginMember1` is used, in this case port 9099.

**Note:** Adding additional transport definitions will not increase the load balancing or failover capabilities of an application server.

### HTTPS considerations in a WLM environment

When using HTTPS transport with a self-signed certificate, you need to distribute client (plug-in) and server (application server) certificates as follows:

- ▶ Import the server certificate from each application server as a trusted CA (certificate authority) into the plug-in's keyfile.
- ▶ If you are using client (plug-in) authentication, you also need to import the client certificate from each plug-in as a trusted CA (certificate authority) into the application server's keyfile.

For details about WebSphere Security, refer to Chapter 10, "Administering WebSphere Security" for information about SSL configuration in the redbook *IBM WebSphere V5.0 Security*, SG24-6573.

### 5.3.3 Creating clusters and cluster members

To allow for workload management and failover, replicas of a Web module are created. This is done by creating cluster members that are part of a cluster. The concepts of clusters and cluster members are covered in 1.3.3, "Workload management using WebSphere clustering" on page 16. Information on how to set up clusters and cluster members can be found in 7.5, "Configuring WebSphere clusters" on page 276.

**Note:** In WebSphere Application Server V5, clustering is performed at the application server level, not the Web module or EJB level.

## **5.4 WebSphere plug-in workload management**

The actual management of Web container workload is performed by the Web server plug-in. This section describes how the plug-in will route requests to the correct Web containers and how the plug-in is configured. Understanding how the plug-in makes these decisions is key to understanding how WebSphere workload manages Web requests.

### **5.4.1 Processing requests**

Once a cluster and its members have been set up, the plug-in can start directing requests from the Web server to the correct application server. Figure 5-7 on page 145 shows what occurs within the plug-in when processing a request.



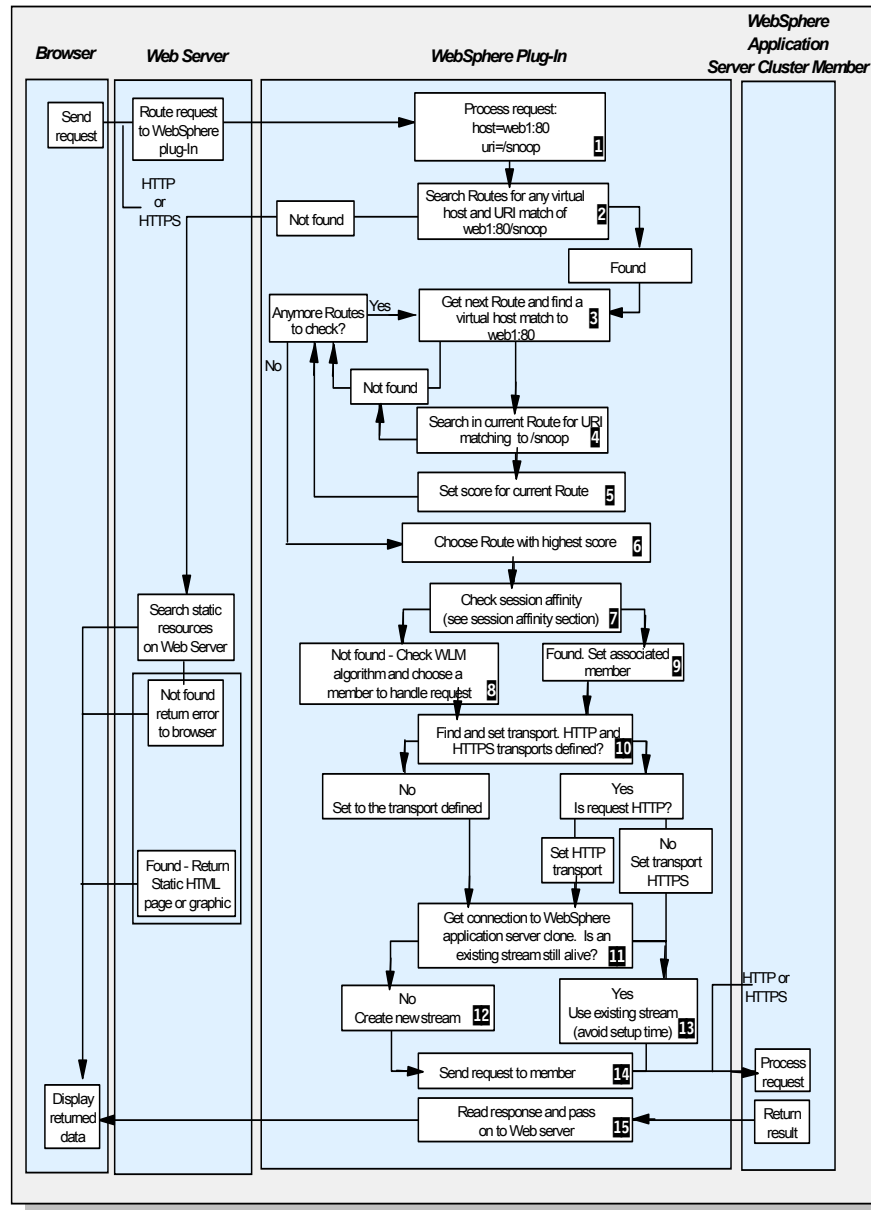


Figure 5-7 How the plug-in processes a request

Here is an example walk-through of Figure 5-7 on page 145. Users asks for the page `http://web1:80/snoop` from their browser. The request is routed to the Web server over the Internet.

1. The Web server immediately passes the request to the plug-in (1). All requests go to the plug-in first.
2. The plug-in then starts by looking at all Route definitions in the `plugin-cfg.xml`. For each Route it searches through its configuration to find if there is any match to the virtual host `web1:80` and URI `/snoop`. It will find the first match and then decide that WebSphere should handle the request (2). If there isn't any match, WebSphere will not handle the request.
3. The plug-in takes the request and separates the host name and port pair and URI. The plug-in now starts by looking at all the defined Routes. It gets a Route and then searches for a virtual host match to `web1` port `80` in that Route. It matches that host name and port to `web1:80` in the `VirtualHost` block (3).
4. The plug-in then tries to match the URI `/snoop` in the current Route. It searches its configuration for a URI mapping that matches the requested URI in the `UriGroup` block (4). It matches `/snoop` in the `UriGroup`.
5. Once the `VirtualHostGroup` and `UriGroup` are found, it sets a score depending on the number of characters in the URI and virtual host (5).
6. The plug-in continues to the next Route and searches through virtual hosts and URI's setting scores for matches with any URI and virtual host match. The Route that has the highest score is chosen (6) and the `ServerCluster` is set.
7. The plug-in now checks the request to see if any session identification has been passed to the server (7). See 5.5, "Session management" on page 166 for more information about this. Our request does not contain any session information.
8. The plug-in chooses a cluster member to manage the request (8). See 5.4.4, "Plug-in workload management and failover policies" on page 156 for more information. A server is chosen to handle the request. If there is a session identifier and a `CloneID` associated with it, the plug-in will choose a server based on the `CloneID` (9).
9. This cluster member has two transports associated to it; HTTP and HTTPS are defined. As this request is HTTP, the cluster member uses the HTTP transport definition (10).
10. In box 11 the term *stream* is used. A stream is a persistent connection to the Web container. Since HTTP 1.1 is used for persistent connections between plug-in and Web container, it is possible to maintain a connection (stream) over a number of requests. In our example, no previous connection has been created from the plug-in to the Web container, so a new stream is created

- (12). If a stream is already established, the plug-in uses the existing stream  
(13).
11. The request is sent to the cluster member (14) and successfully processed.  
The plug-in passes the response on to the Web server (15), which in turn  
passes it back to the user's browser.

## 5.4.2 The plug-in configuration file

Figure 5-7 on page 145 showed the process flow; now let us look at where all the configuration information used in that process is stored.

Example 5-2 shows how the configuration is stored. It is placed in the plugin-cfg.xml file, which is placed by default in the following directory:

<WAS\_HOME>/config/cells

An example is shown in Example 5-2.

*Example 5-2 Plugin-cfg.xml example*

---

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<Config ASDisableNagle="false" AcceptAllContent="false" IISDisableNagle="false"
IgnoreDNSFailures="false" RefreshInterval="60" ResponseChunkSize="64">
  <Log LogLevel="Error" Name="C:\WebSphere\AppServer\logs\http_plugin.log"/>
  <Property Name="ESIEnable" Value="true"/>
  <Property Name="ESIMaxCacheSize" Value="1024"/>
  <Property Name="ESIInvalidationMonitor" Value="false"/>
  <VirtualHostGroup Name="PluginVirtualHost">
    <VirtualHost Name="*:9088"/>
    <VirtualHost Name="web2:80"/>
  </VirtualHostGroup>
  <ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="PluginCluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
    <Server CloneID="v544d031" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="8" MaxConnections="-1" Name="was1node_PluginMember1"
WaitForContinue="false">
      <Transport Hostname="app1.itso.ibm.com" Port="9088" Protocol="http"/>
      <Transport Hostname="app1.itso.ibm.com" Port="9447" Protocol="https">
        <Property Name="keyring"
Value="C:\WebSphere\AppServer\etc\plugin-key.kdb"/>
        <Property Name="stashfile"
Value="C:\WebSphere\AppServer\etc\plugin-key.sth"/>
      </Transport>
    </Server>
    <Server CloneID="v544d0o0" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="was2Node_PluginMember2"
WaitForContinue="false">
```

```

        <Transport Hostname="app2.itso.ibm.com" Port="9086" Protocol="http"/>
        <Transport Hostname="app2.itso.ibm.com" Port="9447" Protocol="https">
            <Property Name="keyring"
Value="C:\WebSphere\AppServer\etc\plugin-key.kdb"/>
            <Property Name="stashfile"
Value="C:\WebSphere\AppServer\etc\plugin-key.sth"/>
        </Transport>
    </Server>
    <PrimaryServers>
        <Server Name="was1node_PluginMember1"/>
        <Server Name="was2Node_PluginMember2"/>
    </PrimaryServers>
</ServerCluster>
    <ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="dmgr_dmManager_Cluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
        <Server ConnectTimeout="0" ExtendedHandshake="false" MaxConnections="-1"
Name="dmManager_dmgr" WaitForContinue="false"/>
        <PrimaryServers>
            <Server Name="dmManager_dmgr"/>
        </PrimaryServers>
    </ServerCluster>
    <UriGroup Name="PluginVirtualHost_PluginCluster_URIs">
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
Name="/snoop/*"/>
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
Name="/hello"/>
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
Name="/hitcount"/>
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
Name="*.jsp"/>
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
Name="*.jsw"/>
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
Name="*.jsw"/>
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
Name="/j_security_check"/>
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
Name="/ibm_security_logout"/>
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
Name="/servlet/*"/>
    </UriGroup>
    <Route ServerCluster="PluginCluster"
UriGroup="PluginVirtualHost_PluginCluster_URIs"
VirtualHostGroup="PluginVirtualHost"/>
    <RequestMetrics armEnabled="false" newBehavior="false" rmEnabled="true"
traceLevel="HOPS">
        <filters enable="true" type="URI">
            <filterValues enable="false" value="/servlet/snoop"/>

```

```

        <filterValues enable="false" value="/webapp/examples/HitCount"/>
        <filterValues enable="true" value="/scenario"/>
    </filters>
    <filters enable="false" type="SOURCE_IP">
        <filterValues enable="false" value="255.255.255.255"/>
        <filterValues enable="false" value="254.254.254.254"/>
    </filters>
</RequestMetrics>
</Config>

```

The tags within this file, listed in Table 5-1, can be associated to the flow chart to help show what each is defining.

*Table 5-1 Plug-in configuration XML tag descriptions*

No.	XML Tag	Description
<b>2,3</b>	VirtualHostGroup VirtualHost	A group of virtual host names and ports that will be specified in the HTTP Host header when the user tries to retrieve a page. Enables you to group virtual host definitions together that are configured to handle similar types of requests. The requested host and port number will be matched to a VirtualHost tag in a VirtualHostGroup.
<b>2,4</b>	UriGroup Uri	A group of URIs that will be specified on the HTTP request line. The incoming client URI will be compared with all the Uri tags in the UriGroup to see if there is match to determine if the application server will handle the request for the Route in conjunction with a virtual host match.

No.	XML Tag	Description
<b>2,3,5,6</b>	Route	<p>The Route definition is the central element of the plug-in configuration. It specifies how the plug-in will handle requests based on certain characteristics of the request. The Route definition contains the other main elements: a required ServerCluster, and either a VirtualHostGroup, UriGroup, or both.</p> <p>Using the information that is defined in the VirtualHostGroup and the UriGroup for the Route, the plug-in determines if the incoming request to the Web server should be sent on to the ServerCluster defined in this Route.</p> <p>The plug-in sets scores for Routes if there is a VirtualHost and Uri match for an incoming request. Once the plug-in processes all Routes, the Route chosen is the one with the highest score.</p>
<b>6</b>	ServerCluster Server	The located ServerCluster from the Route tag contains a list of Server tags that in turn contain the requested object. At <b>7</b> , the Server tag is used to check session affinity. At <b>8</b> or <b>9</b> , the correct server is selected.
<b>8,9</b>	ServerCluster Server	The ServerCluster located by finding the correct Route can optionally specify the WLM algorithm. This will then be used to select one Server from within the ServerGroup.
<b>10</b>	Transport	Once a Server has been located, its Transport tags describe how to connect to it.

**Tip:** A detailed description of each XML tag and its relationships can be found in the WebSphere V5.1 InfoCenter at

<http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp>

To find the appropriate section, select the **WebSphere Application Server Network Deployment** package in the upper left corner. Then select **Administering -> Task overviews -> Configuring Web server plug-ins -> plugin-cfg.xml file**.

There are some settings in the plug-in file that can affect how the plug-in works in a workload management environment:

- ▶ It is possible to change the policy for workload distribution in the configuration file. See 5.4.4, “Plug-in workload management and failover policies” on page 156.
- ▶ In the ServerCluster tag, it is possible to change the retry interval for connecting to a cluster member marked as down.
- ▶ When cluster members are set up, it is possible to affect whether session management processing will occur for a request. If you are not using sessions, remove the CloneID from all the server tags within a ServerCluster. The plug-in will then bypass session management processing, increasing performance.
- ▶ It is possible to change a refresh interval for the XML file. This is done in the Config tag. This defines how often the plug-in will check to see if the configuration file has changed.
- ▶ It is possible to change the maximum number of connections that will be allowed to a server from a given plug-in. This is done in the Server tag. By default, MaxConnections is set to -1. If this attribute is set to either zero or -1, there is no limit to the number of pending connections to the Application Servers.

### 5.4.3 Generation of the plug-in configuration file

Generating the plug-in configuration recreates the plugin-cfg.xml using the newest settings of objects in the cell. Once the plug-in has reloaded its new configuration, clients are able to access the updated or new Web resources.

**Note:** When generating the plugin-cfg.xml file on any running node other than the Deployment Manager, the plugin-cfg.xml will be a subset of the cell-wide configuration. This is the result of each node only having a subset of the master configuration repository locally on the node.

When regenerating, there will be a delay in the changes taking effect on your Web site. This delay is governed by two settings:

- ▶ Auto-reload setting of your Web module
- ▶ Refresh interval of the plug-in

## Auto-reload setting of your Web module

The first part of the delay is the auto-reload setting of the Web module. This can be defined using the following methods:

- ▶ Application Server Toolkit (ASTK)

Open the deployment descriptor editor, and then click the **IBM Extensions** tab. Here one can set whether reload is enabled and how often reloading should occur.

- ▶ Manual editing of non-deployed application

The auto-reload settings (reloadInterval and reloadEnabled) can be changed by manually editing the ibm-web-ext.xmi deployment descriptor file in the WEB-INF directory of the Web module.

- ▶ Manual editing of already deployed application

The auto-reload settings of an already deployed application can be changed by manually editing the ibm-web-ext.xmi deployment descriptor in the WEB-INF directory of the deployed Web module.

As a result of the application dynamic reloading feature provided in WebSphere 5.1, manual changes to the above deployment descriptor will be automatically detected and loaded into the runtime, except for the following cases:

- Change reloadEnabled setting (true or false)
- Set reloadInterval setting to zero

When reloading is enabled, WebSphere monitors the classpath of the Web application and whenever a component update is detected, all components (JAR or class files) are reloaded.

The longer the reload interval, the longer it will take for the Web module to use the new components. Although new information will appear in the plugin-cfg.xml file, updated Web module components are not used until they have been reloaded.

## Refresh interval of the plug-in

There will also be a delay in the WebSphere plug-in, since it checks for a new configuration file only at specified time intervals. This interval (in seconds) is determined by the refreshInterval attribute of the Config element in the plugin-cfg.xml file. If a new configuration file has been generated, then there could be a delay of anything up to the number specified in this setting before the plug-in will load the new configuration.

*Example 5-3 Plugin-cfg.xml extract showing RefreshInterval*

---

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```



```
<Config ASDisableNagle="false" AcceptAllContent="false" IISDisableNagle="false"
IgnoreDNSFailures="false" RefreshInterval="60" ResponseChunkSize="64">
  <Log LogLevel="Error" Name="C:\WebSphere\AppServer\logs\http_plugin.log"/>
```

---

If the RefreshInterval has passed, the actual loading of the new plug-in configuration file into the plug-in runtime is triggered by a request to the Web server.

In a development environment in which changes are frequent, a lower setting than the default setting of 60 is preferable. In production, a higher value than the default is preferable because updates to the configuration will not occur as often.

**Note:** There are two situations where the generated plug-in configuration file will not reflect changes to an application without the application server(s) being restarted:

1. If a change is made to the virtual host that a Web module uses.
2. If auto-reload is disabled in a Web module. Upon regeneration, the new URI information will still appear in the plugin-cfg.xml file, but the updated Web module components will not be picked up until the application server is restarted.

WebSphere Application Server V5.1 does not automatically regenerate the plugin-cfg.xml file. There are also certain settings in the plug-in configuration file that cannot be configured via WebSphere administrative tools, for example the load balancing method or the retry interval. These values need to be set manually.

If you are working in a development environment, where constant changes are being made to installed applications, you could use the embedded HTTP transport to access your application.

For example, point your browser to `http://myhost:9080/servlet`, which will bypass the plug-in.

## Manual regeneration of the plug-in file

Manual regeneration of the plug-in file can be performed at any point, whether application servers are running or not. The result and delay are discussed in 5.4.3, “Generation of the plug-in configuration file” on page 151. There are two methods of manual regeneration:

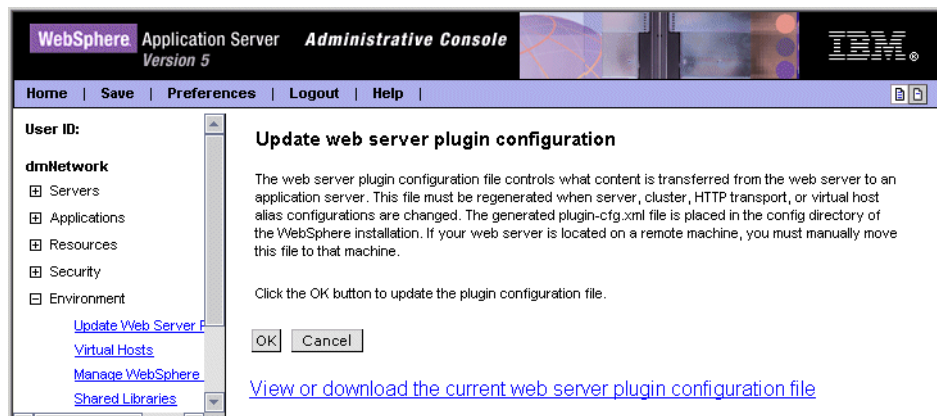
- ▶ From the Administrative Console
- ▶ Via a command line using the **GenPluginCfg** command

### ***From the Administrative Console***

The Administrative Console can be used to regenerate the plug-in. In a distributed environment, the plug-in will only be regenerated on the Deployment Manager system. This means that all the directory paths of the Deployment Manager will be used, so when pushing the plugin-cfg.xml file to other systems make sure that the paths in the plugin-cfg.xml correspond to the paths of the machine you have copied it to.

To regenerate the plug-in file manually from the Administrative Console:

1. Select **Environment -> Update Web Server plug-in** as shown in Figure 5-8.
2. Click **OK**. A message will appear to say whether the plug-in regeneration was successful.



*Figure 5-8 Regenerating the plug-in on the Deployment Manager*

3. Select **System Administration -> Nodes** as shown in Figure 5-9 on page 155.
4. Select the nodes you want to push the plugin-cfg.xml file to.
5. If your Web servers run at the same nodes as the application servers, click **Full Resynchronize**. If the Web servers run at remote nodes from the application servers, copy the plugin-cfg.xml to the Web server nodes.

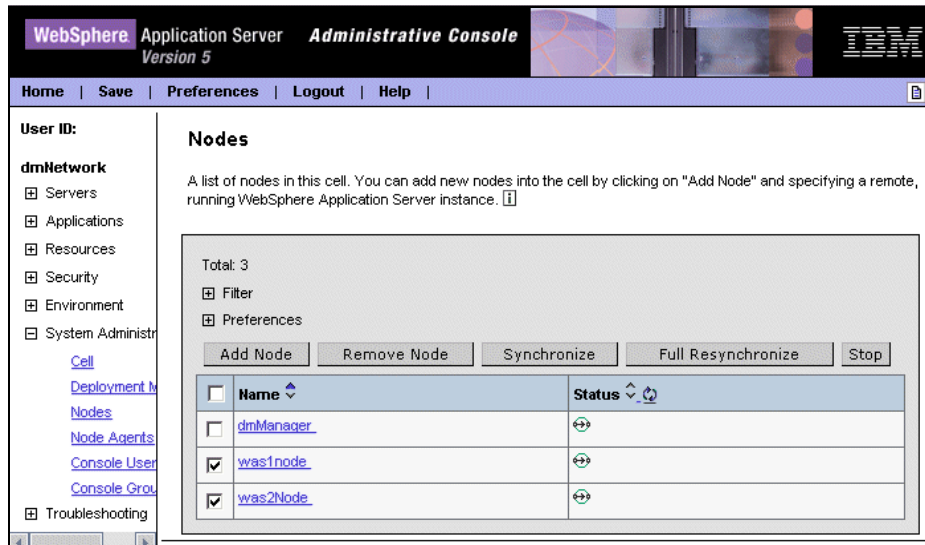


Figure 5-9 Full Resynchronization

### GenPluginCfg command

The command **GenPluginCfg** can be used in its simplest form by typing:

```
GenPluginCfg.sh
```

Or,

```
GenPluginCfg.bat (for Windows environments)
```

This will generate the plug-in configuration file on the node you have run the command on. However, it is recommended that you run this command on the Deployment Manager node and then copy the file to all other nodes to make sure the complete environment's configuration is taken into account. If your Web servers run at the same nodes as your application servers, the command **syncNode** can be used for copying the plug-in configuration file to the nodes. However, this command can be used only when the Node Agents are stopped.

When generating the plug-in on a node other than the Deployment Manager node, the plug-in will only be regenerated with a subset of the configuration that is local to the node you have run the command from. Therefore generating the plug-in on the Deployment Manager node is the most convenient form of generating the plug-in.

There are many other command-line tools available in WebSphere Application Server V5.1. For details on the **GenPluginCfg** and **syncNode** commands as well as other available commands, refer to Appendix A, "Command-line tools", of the

## 5.4.4 Plug-in workload management and failover policies

This section describes the process that the plug-in goes through to choose a cluster member to serve a request. We cover the algorithms used to choose a cluster member and the various ways this process can be manipulated. We also take a deeper look at how the plug-in connects to each application server. With this knowledge, it is possible to understand how failover situations can arise and what will happen.

### Workload management policies

In WebSphere Application Server V5.1, the plug-in has two options for the load distributing algorithm:

- ▶ Round robin with weighting
- ▶ Random

**Note:** The weighting for the round robin approach can be turned off by giving the application servers equal weights.

The default value is Round Robin. It can be changed by editing the plugin-cfg.xml file and amend each ServerCluster tag to:

```
<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="PluginCluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
```

or:

```
<ServerCluster CloneSeparatorChange="false" LoadBalance="Random"
Name="PluginCluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
```

There is also a feature in WebSphere Application Server V5.1 for the plug-in called ClusterAddress that can be used to suppress load balancing. See “ClusterAddress” on page 159 for more details.

For examples of how to see these workload management policies in action, refer to 5.7.1, “Normal operation” on page 185.

### **Weighted round robin for WebSphere Application Server V5.1**

When using this algorithm, the plug-in selects a cluster member at random from which to start. The first successful browser request will be routed to this cluster member and then its weight is decremented by 1. New browser requests are

then sent round robin to the other application servers and subsequently the weight for each application server is decremented by 1. The spreading of the load is equal between application servers until one application server reaches a weight of 0. From then on, only application servers without a weight of 0 will be routed requests to. The only exception to this pattern is when a cluster member is added or restarted.

**Notes:**

- ▶ The first request goes to a random application server to avoid multiple Web servers and/or multi-process Web servers directing initial requests to the same application server.
- ▶ When starting the HTTP Server, the application server weight is reduced to the lowest common denominator. For example: PluginMember1's weight is 8 and PluginMember2's weight is 6. When you start the HTTP Server, the weight of PluginMember1 will be set to 4 and the weight of PluginMember2 will be set to 3.

Using Table 5-2 on page 158 and the following explanations will show you how weighted round robin is performed. To begin with, we have a weight of 4 for PluginMember1 and a weight of 3 for PluginMember2:

1. When the first request comes in, PluginMember1 is randomly picked. The request is OK. PluginMember's weight is decremented by 1 to 3.
2. The second request is sent to PluginMember2. The request is OK. PluginMember2's weight is decremented by 1 to 2.
3. The third and fourth requests are sent to PluginMember1 and PluginMember2 respectively. So PluginMember1 now has a weight of 2 and PluginMember2 now has a weight of 1.
4. The fifth request has a cookie that specifies a server affinity to PluginMember1. The request is sent to PluginMember1 and its weight is decremented by 1 to 1.
5. The sixth request is again sent to PluginMember1 because of server affinity. The request is OK. PluginMember1's weight is decremented by 1 to 0.
6. The seventh request again has a cookie that specifies server affinity to PluginMember1. The request is sent to PluginMember1 and its weight is decremented by 1 to -1.
7. The eighth to eleventh request all have server affinity to PluginMember1. The weight is decremented by 1 for each request. After the eleventh request, PluginMember1 now has a weight of -5 while PluginMember2 still has a weight of 1.

8. The twelfth request has no server affinity so it is sent to PluginMember2. PluginMember2's weight is decremented by 1 to 0.
9. When processing the thirteenth request, the plug-in decides to reset the weights because there are no servers marked up having positive weights. A multiple of the lowest common denominator of the servers' maximum weight is added back to the current weights to make all weights positive again. See the Important box below for a detailed description on how the weights are reset.

After resetting the weights, the sequence gets repeated with the same starting point (no random server selection this time), in our case, this means that the thirteenth request is sent to PluginMember1 (after the weights have been reset) and PluginMember1's weight is decremented by 1 to 2.

**Important:** In our example the current weight of PluginMember1 is -5 because many session-based requests have been served. PluginMember2 has a weight of 0. The plug-in checks how many times the maxWeight should be added to make the current weight positive. The starting weight for PluginMember1 was 4 and 3 for PluginMember2. Because PluginMember1's current weight is -5, adding 4 (the lowest common denominator) would not set it to a positive weight. Thus the plug-in decides to add the starting weights \* 2, which is 8 for PluginMember1 and 6 for PluginMember2. So the new current weights are 3 for PluginMember1 ( $-5 + 2 * 4$ ) and 6 for PluginMember2 ( $0 + 2 * 3$ ).

Table 5-2 Request handling using weighted round robin server selection

Number of Requests	PluginMember1 Weight	PluginMember2 Weight
0	4	3
1	3	3
2	3	2
3	2	2
4	2	1
5	1	1
6 - Request with session affinity to PluginMember1	0	1
7 - Request with session affinity to PluginMember1	-1	1

Number of Requests	PluginMember1 Weight	PluginMember2 Weight
8 - Request with session affinity to PluginMember1	-2	1
9 - Request with session affinity to PluginMember1	-3	1
10 - Request with session affinity to PluginMember1	-4	1
11 - Request with session affinity to PluginMember1	-5	1
12 - No session affinity for this request	-5	0
<b>13 RESET - no session affinity for this request</b>	2	6

### ***Random***

Requests are passed to cluster members randomly. Weights are not taken into account as per round robin. The only time the application servers are not chosen randomly is when there are requests with sessions associated with them. When the random setting is used, cluster member selection does not take into account where the last request was handled. This means that a new request could be handled by the same cluster member as the last request.

### ***ClusterAddress***

As mentioned earlier, the plugin-cfg.xml tag called ClusterAddress can be used to suppress plug-in based load balancing. So you define a ClusterAddress when you do *not* want the plug-in to perform any type of load balancing because you already have some type of load balancer in between the plug-in and the application servers — which can be a software or a hardware solution. The ClusterAddress specified is the IP address of your external load balancing solution. When doing so, the plug-in will only focus on route determination and session affinity. Once ClusterAddress is defined, if a request comes in that does not have session affinity established, the plug-in routes it to the ClusterAddress. If affinity has been established, then the plug-in routes the request directly to the appropriate application server, bypassing the ClusterAddress entirely. If no ClusterAddress is defined for the ServerCluster, then the plug-in load balances across the PrimaryServers list.

The ClusterAddress tag has the same attributes and elements as a Server element. The difference is that you can only define one of them within a ServerCluster. An example of how to configure the plugin-cfg.xml is shown in Example 5-4 on page 160.

*Example 5-4 Extract of plugin-cfg.xml showing ClusterAddress*

---

```
.
<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="PluginCluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
  <ClusterAddress Name="LoadBalancer">
    <Transport Hostname="lb.itso.ibm.com" Port="80" Protocol="http"/>
  </ClusterAddress>
  <Server CloneID="v544d031" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="8" MaxConnections="-1" Name="was1node_PluginMember1"
WaitForContinue="false">
    <Transport Hostname="app1.itso.ibm.com" Port="9088" Protocol="http"/>
  </Server>
  <Server CloneID="v544d0o0" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="was2Node_PluginMember2"
WaitForContinue="false">
    <Transport Hostname="app2.itso.ibm.com" Port="9086" Protocol="http"/>
  </Server>
  <PrimaryServers>
    <Server Name="was1node_PluginMember1"/>
    <Server Name="was2Node_PluginMember2"/>
  </PrimaryServers>
</ServerCluster>
.
```

---

A plug-in trace showing how the plug-in handles the ClusterAddress element is shown in Example 5-5 on page 161, using the plugin-cfg.xml shown in Example 5-4.



#### *Example 5-5 Plug-in trace showing the ClusterAddress functionality*

---

```
ws_common: websphereHandleSessionAffinity: Checking for session affinity
ws_common: websphereHandleSessionAffinity: Checking the SSL session id
lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'SSLJSESSION'
lib_htrequest: htrequestGetCookieValue: No cookie found for: 'SSLJSESSION'
ws_common: websphereHandleSessionAffinity: Checking the cookie affinity:
JSESSIONID
lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
lib_htrequest: htrequestGetCookieValue: No cookie found for: 'JSESSIONID'
ws_common: websphereHandleSessionAffinity: Checking the url rewrite affinity:
jsessionid
ws_common: websphereParseSessionID: Parsing session id from '/snoop'
ws_common: websphereParseSessionID: Failed to parse session id
ws_common: websphereFindServer: Have a cluster address server so using it:
LoadBalancer
ws_common: websphereFindTransport: Finding the transport
ws_common: websphereFindTransport: Setting the transport(case 2):
lb.itso.ibm.com on port 80
ws_common: websphereExecute: Executing the transaction with the app server
ws_common: websphereGetStream: Getting the stream to the app server
```

---

### **Failover**

There are a number of situations when the plug-in might not be able to complete a request to an application server. In a clustered environment with several cluster members this does not need to interrupt service.

Here are some example situations when the plug-in cannot connect to a cluster member:

- ▶ The cluster member has been brought down intentionally for maintenance.
- ▶ The application server crashes.
- ▶ There is a network problem between the plug-in and the cluster member.
- ▶ The cluster member is overloaded and cannot process the request.

When the plug-in has selected a cluster member to handle a request (see Figure 5-7 on page 145, boxes 7, 8 and 9), it will attempt to communicate with the cluster member. If this communication is unsuccessful or breaks, then the plug-in will mark the cluster member as down and attempt to find another cluster member to handle the request.

The marking of the cluster member as down means that, should that cluster member be chosen as part of a workload management policy or in session affinity, the plug-in will not try to connect to it. The plug-in will see that it is marked as down and ignore it.

The plug-in will wait for a period of time before removing the marked as down status from the cluster member. This period of time is called the *retry interval*. By default, the retry interval is 60 seconds. If you turn on tracing in the plug-in log file, it is possible to see how long is left until the cluster member will be tried again.

To set `RetryInterval`, edit the `plugin-cfg.xml` file and locate the `ServerCluster` tag for the required server cluster. The value can be any integer. Below is an example of this:

```
<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="PluginCluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="10">
```

This sets the `RetryInterval` to 10 seconds. If a downed cluster member in this server cluster is selected for a request, and 10 seconds have elapsed since the last attempt, the plug-in will try again to connect to it.

By marking the cluster member as down, the plug-in does not spend time at every request attempting to connect to it again. It will continue using other available cluster members without retrying the down cluster member, until the retry interval has elapsed.

Figure 5-10 on page 163 shows how this selection process works. It is an expansion of box 8 from Figure 5-7 on page 145.

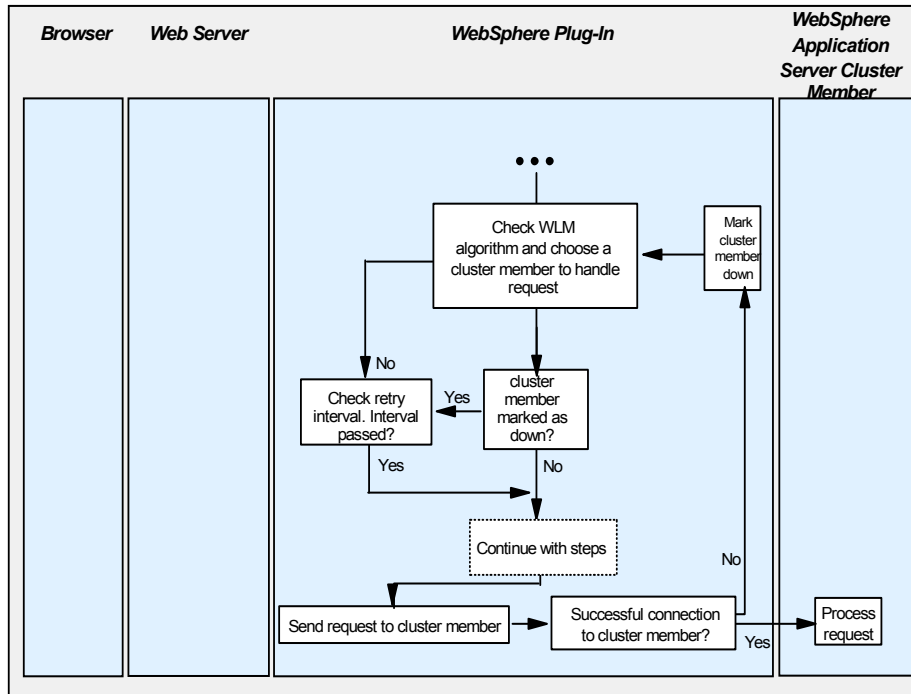


Figure 5-10 Failover selection process

**Note:** For more information about failover, go to 5.7, “Web server plug-in behavior and failover” on page 184. It runs through some examples of what happens in a failover situation.

## Primary and backup servers

WebSphere Application Server V5.1 introduces a new plug-in feature called primary and backup servers. The `plugin-cfg.xml` when generated will have a tag called *PrimaryServers*, which is an ordered list of servers to which to send requests. There is also an optional tag called *BackupServers*. This is an ordered list of servers to which requests should be sent to if all servers specified in the *PrimaryServers* list are unavailable. The plug-in does not load balance across the *BackupServers* list but traverses the list in order until no servers are left in the list or until a request is successfully sent and a response received from an application server.

**Note:** The `plugin-cfg.xml` file must be edited manually to add backup servers.

All application server details are listed in `ServerCluster` with the primary and backup server list being used as a pointer to those hosts that you want to specify as backup or primary servers. This is illustrated in Example 5-6.

*Example 5-6 ServerCluster element depicting primary and backup servers.*

---

```
.
<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="PluginCluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
  <Server CloneID="v544d031" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="8" MaxConnections="-1" Name="was1node_PluginMember1"
WaitForContinue="false">
    <Transport Hostname="app1.itso.ibm.com" Port="9088" Protocol="http"/>
    <Transport Hostname="app1.itso.ibm.com" Port="9447" Protocol="https">
      <Property Name="keyring"
Value="C:\WebSphere\AppServer\etc\plugin-key.kdb"/>
      <Property Name="stashfile"
Value="C:\WebSphere\AppServer\etc\plugin-key.sth"/>
    </Transport>
  </Server>
  <Server CloneID="v544d0o0" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="was2Node_PluginMember2"
WaitForContinue="false">
    <Transport Hostname="app2.itso.ibm.com" Port="9086" Protocol="http"/>
    <Transport Hostname="app2.itso.ibm.com" Port="9447" Protocol="https">
      <Property Name="keyring"
Value="C:\WebSphere\AppServer\etc\plugin-key.kdb"/>
      <Property Name="stashfile"
Value="C:\WebSphere\AppServer\etc\plugin-key.sth"/>
    </Transport>
  </Server>
  <Server CloneID="v54deevk" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="was1node_PluginMember3"
WaitForContinue="false">
    <Transport Hostname="app1.itso.ibm.com" Port="9093" Protocol="http"/>
    <Transport Hostname="app1.itso.ibm.com" Port="9448" Protocol="https">
      <Property Name="keyring"
Value="C:\WebSphere\AppServer\etc\plugin-key.kdb"/>
      <Property Name="stashfile"
Value="C:\WebSphere\AppServer\etc\plugin-key.sth"/>
    </Transport>
  </Server>
  <Server CloneID="v54defqf" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="was2Node_PluginMember4"
WaitForContinue="false">
    <Transport Hostname="app2.itso.ibm.com" Port="9089" Protocol="http"/>
    <Transport Hostname="app2.itso.ibm.com" Port="9448" Protocol="https">
```

```

        <Property Name="keyring"
Value="C:\WebSphere\AppServer\etc\plugin-key.kdb"/>
        <Property Name="stashfile"
Value="C:\WebSphere\AppServer\etc\plugin-key.sth"/>
    </Transport>
</Server>
<PrimaryServers>
    <Server Name="was1node_PluginMember1"/>
    <Server Name="was2Node_PluginMember2"/>
</PrimaryServers>
<BackupServers>
    <Server Name="was1node_PluginMember3"/>
    <Server Name="was2Node_PluginMember4"/>
</BackupServers>
</ServerCluster>

```

The backup server list is only used when all primary servers are down. Figure 5-11 shows this process in detail.

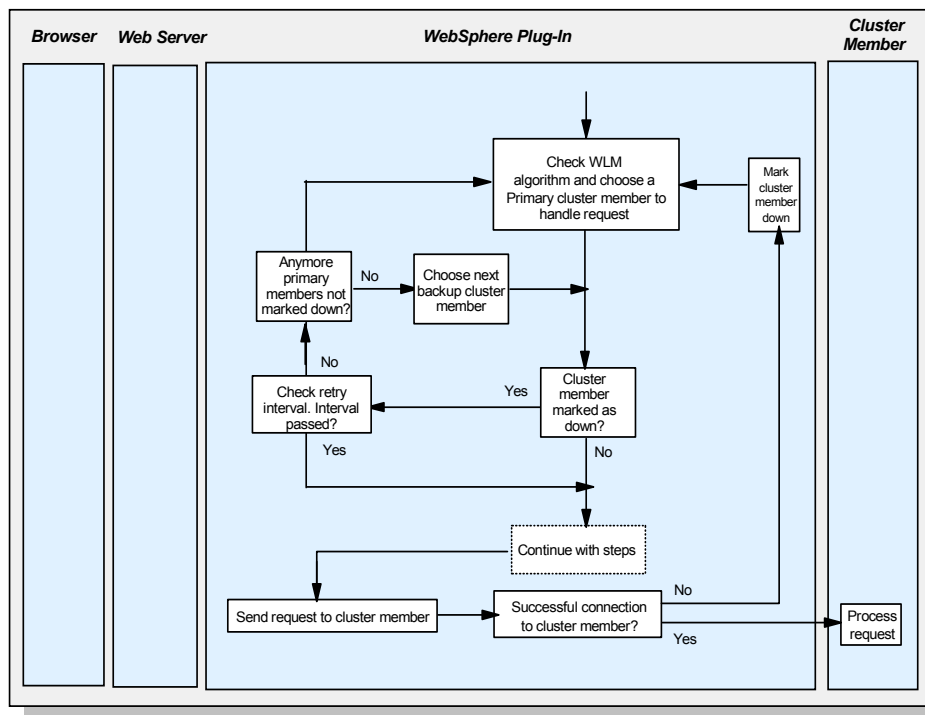


Figure 5-11 Primary and Backup server selection process

1. The request comes in and is sent to the plug-in.
2. The plug-in chooses the next primary server, checks whether the cluster member has been marked down and the retry interval. If it has been marked down and the retry timer is not 0, it continues to the next cluster member.
3. A stream is opened to the application server (if not already open) and a connection is attempted.
4. The connection to the cluster member fails. When a connection to a cluster member has failed, the process begins again.
5. The plug-in repeats steps 2, 3 and 4 until all primary servers are marked down.
6. When all primary servers are marked as down, the plug-in will then repeat steps 2 and 3 with the backup server list.
7. It performs steps 2 and 3 with a successful connection to the backup server. Data is sent to the Web container and is then returned to the user. If the connection is unsuccessful, the plug-in will then go through all the primary servers again and then all the servers marked down in the backup server list until it reaches a backup server that is not marked down.
8. If another request comes in and one of the primary server's retry timer is at 0 the plug-in will try and connect to it.

## 5.5 Session management

One of the other functions that the plug-in provides, in addition to failover and workload management, is the ability to manage session affinity.

HTTP sessions allow applications to maintain state information across multiple user visits. The Java servlet specification provides a mechanism for applications to maintain all session information at the server and exchange minimal information with a user's browser.

The methods used to manage the sessions are very important in a workload-managed environment. There are different methods of identifying, maintaining affinity, and persisting the sessions.

See "HTTP sessions and the session management facility" on page 22 for basic information or Chapter 14, "Configuring session management", of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195 for details on this subject.

### 5.5.1 Session affinity

In the Servlet 2.3 specification, as implemented by WebSphere Application Server V5.0 and higher, only a single cluster member may control/access a given session at a time. After a session has been created, all requests need to go to the application server that created the session. This session affinity is provided by the plug-in.

In a clustered environment, there will be more than one application server that can serve the client request. The plug-in needs to read a request and be able to identify which cluster member should handle it. Session identifiers are used to do this. The session key is used by the Web container to retrieve the current session object.

*Example 5-7 Example of a session identifier*

---

JSESSIONID=0000A2MB4IJozU\_VM8IffsMNfdR:v544d0o0

---

As shown in Example 5-7, the JSESSIONID cookie can be divided into four parts:

- ▶ Cache ID
- ▶ Session ID
- ▶ Separator
- ▶ Clone ID

Table 5-3 shows the mappings of these parts based on Example 5-7.

*Table 5-3 JSESSIONID content*

Content	Value used in the example
Cache ID	0000
Session ID	A2MB4IJozU_VM8IffsMNfdR
Separator	:
Clone ID (cluster member)	v544d0o0

A clone ID is the ID of a cluster member as shown in Example 5-8.

*Example 5-8 Extract of plugin-cfg.xml showing the CloneID*

---

```
<Server CloneID="v544d0o0" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="was2Node_PluginMember2"
WaitForContinue="false">
```

---

## 5.5.2 Session failover inside the plug-in

If the server that contains the session is not available to the plug-in when it forwards the request, then the user will not be able to continue the session. To overcome this problem, it is possible to create a persistent store for the session information. This means that if a server is unavailable, the plug-in can route the request to an alternate server as illustrated in Figure 5-11 on page 165. The alternate server can then retrieve the session information from the persistent store.

There are two mechanisms to configure session persistence in WebSphere Application Server V5.1:

- ▶ Database persistence

Session state is persisted to a database shared between the clustered application servers. This feature was the only session persistence mechanism provided by earlier versions of WebSphere Application Server.

- ▶ Memory-to-memory replication, also known as WebSphere Internal Messaging

This is a new feature provided since WebSphere 5.0. It provides in-memory replication of session state between clustered application servers.



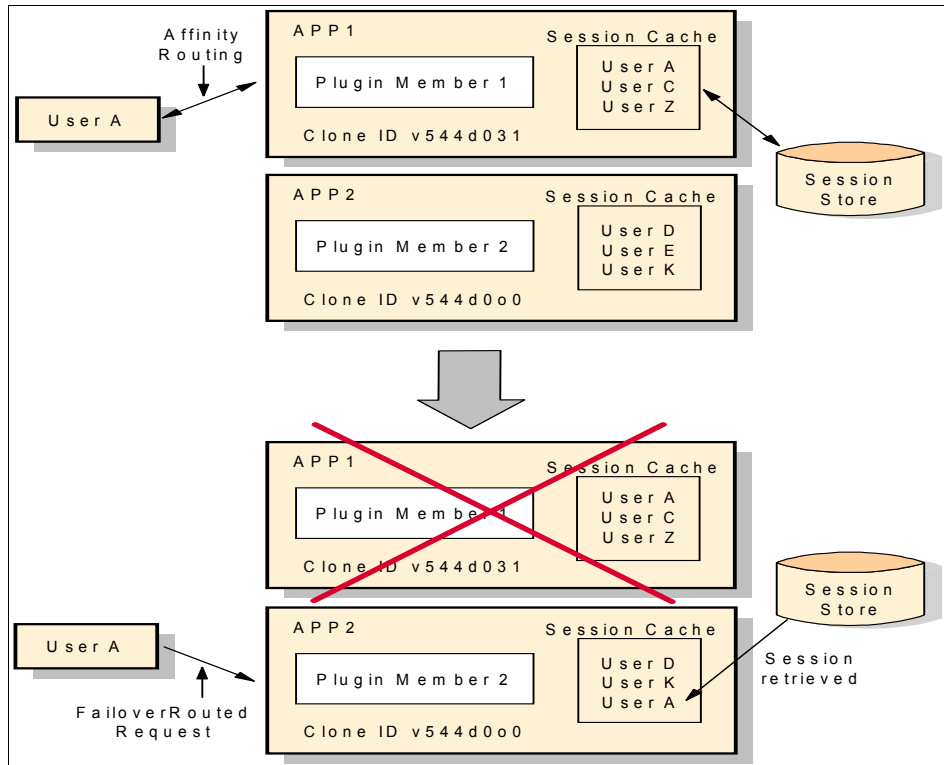


Figure 5-12 Session management sample topology

Using Figure 5-12, the steps involved using our sample topology are:

1. The plug-in processes a request from user A to `http://yourWebServer/snoop`. The request also contains a JSESSION cookie with a session ID and CloneID of v544d031.
2. Plug-in matches the virtual host and URI to PluginCluster.
3. The plug-in then checks for session affinity and finds the CloneID of v544d031 in the requests JSESSION cookie.
4. It then searches for the CloneID of v544d031 in the plug-cfg.xml's list of primary servers and matches the CloneID to PluginMember1.
5. The plug-in will then check to see if PluginMember1 has been marked down. In our case it has not been marked down.
6. It will then attempt to get a stream to PluginMember1. Finding the server is not responding, PluginMember1 is marked as down and the retry timer is started.
7. The plug-in then checks the session identifier again.

8. It then checks the servers. When it reaches PluginMember1, it finds it is marked down and the retry timer is not 0, so it skips PluginMember1 and checks the next cluster member in the primary list.
9. PluginMember2 (CloneID v544d0o0) is selected and the plug-in attempts to get a stream to it. The plug-in either opens a stream or gets an existing one from the queue.
10. The request is sent and received successfully to PluginMember2 and sent back to user A.

### 5.5.3 Session identifiers

There are three methods of identifying a user's session to the application server. They are:

- ▶ Cookies
- ▶ URL rewriting
- ▶ SSL ID

In WebSphere Application Server V5.x, session management can be defined at the following levels:

- ▶ Application server

This is the default level. Configuration at this level is applied to all Web modules within the server.

- ▶ Application

Configuration at this level is applied to all Web modules within the application.

- ▶ Web module

Configuration at this level is applied only to that Web module.

See Chapter 14, "Configuring session management", of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195 on how to configure sessions.

#### Cookies

When this option is selected from the session manager pane, the plug-in will use the JSESSIONID to manage requests. This name is required by the Servlet 2.3 specification. The session management tracking mechanism can be performed at the application server, Web container, or the Web module level. To learn more about this, refer to Chapter 14, "Configuring session management" of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195.

To use the cookie JSESSIONID, users must have their browsers set up to allow cookies.

None of the workload management issues discussed in “SSL ID” on page 172 applies to cookies. The browser can be connected to any Web server and there will be no effect on the session. Cookie session identifiers will survive a Web server crash and, provided persistent sessions are enabled, will also survive unavailability of the application server.

The session lifetime is governed by the cookie lifespan. By default, WebSphere defines its cookies to last until the browser is closed. It is also possible to define the maximum age of the cookie in the cookies configuration.

### ***When to use cookies***

Cookies are the most common method of tracking the session. They work well in a workload-managed environment, with ease-of-use advantages over SSL IDs.

They do not require additional application coding and can be used from static HTML links, unlike URL rewriting.

The fact that any user can turn off cookie support in his/her browser could be more important to your application. If this is the case, then one of the other options must be considered.

If security is important, then it is possible to use cookies in an SSL environment and not have the overhead of setting up SSL session ID management.

### **URL rewriting**

URL rewriting (or URL encoding) is a useful mechanism that does not require users to enable cookies in their browsers, and yet still allows WebSphere to manage sessions.

The process of setting up URL rewriting is not transparent to the Web application. It requires a developer to include specific commands to append the session information to the end of any HTTP link that will be used from the Web browser. The plug-in will search for any URL encoded session information about incoming requests and route them accordingly.

Rewriting the URLs can only be done on dynamic HTML that is generated within WebSphere, for example the output from JSPs or servlets. Session information will be lost if static HTML links are accessed, restricting the flow of site pages to dynamic pages only. From the first page, the user receives a session ID, and the Web site must continue using dynamic pages until the completion of the session.

There are no specific issues with using URL encoding in a workload-managed environment.

### ***When to use URL encoding***

Due to the restrictions mentioned above, the only situation in which URL encoding excels over the other options is when users have not enabled cookies in their browsers.

Because it is possible to select more than one mechanism to pass session IDs, it is also possible to compensate for users not using cookies. URL encoding could be enabled and then used as a fallback mechanism if the users are not accepting cookies.

### **SSL ID**

To use the SSL ID as the session modifier, clients need to be using an SSL connection to the Web server. This connection does not need to use client certificate authentication, but simply a normal server authentication connection. This can be enabled by turning on SSL support in the Web server. The steps for doing this are detailed in the redbook *IBM WebSphere V5.0 Security*, SG24-6573.

The session ID is generated from the SSL session information. This is passed to the Web server and then passed on to the plug-in. If more than one Web server is being used, then affinity must be maintained to the correct Web server, since the session ID is defined by the connection between browser and Web server. Connecting to another Web server will reset the SSL connection and a new session ID will be generated.

SSL tracking is supported only for the IBM HTTP Server and SUN ONE Web Server.

It is possible to maintain the SSL session affinity using the Load Balancer from IBM WebSphere Edge Components. See 4.5.5, “SSL session ID” on page 120 for details.

SSL session ID cannot be used on its own in a clustered environment. It is not possible to add the cluster member ID to the end of the SSL session information, so another mechanism must be used. Either cookies or URL rewriting needs to be enabled to provide this function. The cookie or rewritten URL then contains session affinity information that enables the Web server to properly route requests back to the same server once the HTTP session has been created on a server. If cookies or URL rewriting are not enabled, then a session will be created but there will be no mechanism to return the user to the correct cluster member at their next request.

The format of the cookie or URL rewrite is shown in Example 5-9 on page 173.

SSLJSESSION=0000SESSIONMANAGEMENTAFFINI:v544d0o0

---

This is the same format as described in Example 5-7 on page 167 but in place of the session ID is the word SESSIONMANAGEMENTAFFINI.

With SSL, the session timeout is not controlled by the application server. The session timeout delay is governed by the Web server and the Web browser. The lifetime of an SSL session ID can be controlled by configuration options in the Web server.

### ***When to use SSL ID***

When using a clustered environment, SSL ID requires a lot of overhead to set up the infrastructure. There is also a single point of failure for each session: the Web server. If the Web server goes down, the user will lose the session ID and therefore access to session information.

SSL ID will also be slower than other mechanisms, since the browser has to communicate using HTTPS and not HTTP. The inconsistency of browsers and their SSL handling could also affect how the application performs.

However, SSL provides a more secure mechanism of user identification for session information. The session identifier is difficult to copy.

If your Web site requires the highest security, then use SSL ID, but be aware that it comes with the overheads and deficiencies mentioned above. Consider using standard cookies over an SSL session instead.

## **5.5.4 Session persistence and failover**

Once an application server has established a session for a user, the user's subsequent requests must be sent to the same application server. The session is stored in a local cache on the application server, as defined in the settings of the session manager, shown in Figure 5-13 on page 174.

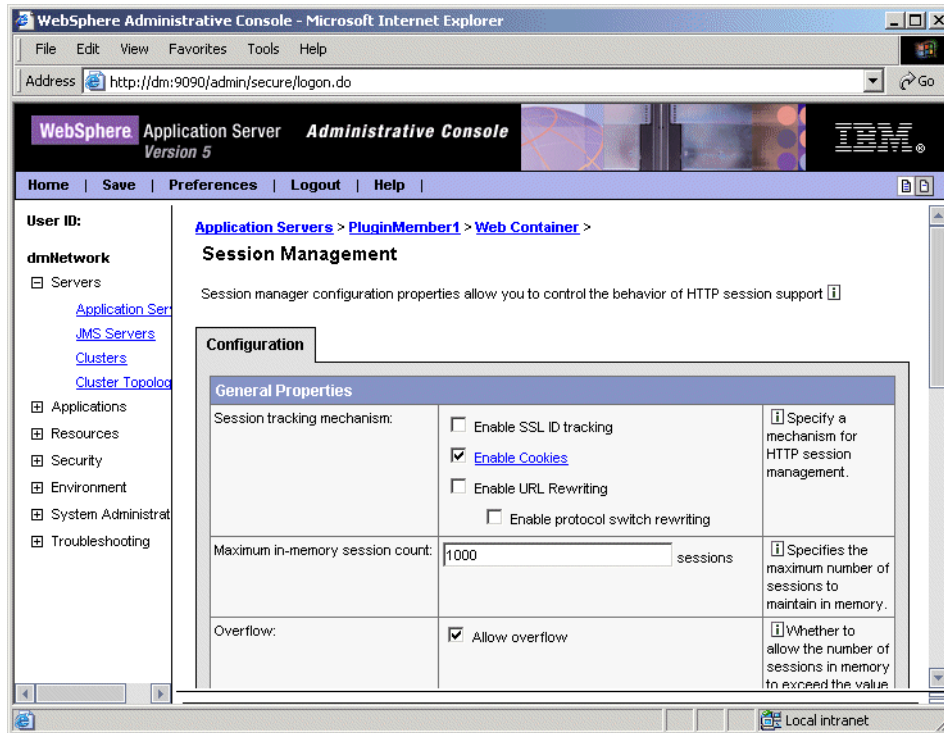


Figure 5-13 Local session cache settings

If this application server is unavailable when the plug-in attempts to connect to it, the plug-in will choose another cluster member and attempt connection. Once a connection to a cluster member is successful, the session manager will decide what to do with the session, as shown in Figure 5-10 on page 163.

The cluster member will find that it does not have the session cached locally and thus will create a new session.

To avoid the creation of a new session, a persistent database or memory-to-memory replication can be used to access sessions from other application servers. Persistent stores use a database to hold the session information from all cluster members. Memory-to-memory replication is a new feature of WebSphere V5.0. It enables the sharing of sessions between application servers without using a database.

## Database session management

The use of a persistent store for sessions is not limited to use in a failover situation. It can also be used when an administrator requires greater control over the session cache memory usage.

In this book, we cover the basic setup and settings that are important to workload management and failover. Chapter 14, “Configuring session management”, of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195, provides a very detailed explanation of the advanced setup of the session database and the effects of the various settings.

### ***Enable persistent sessions***

Follow these steps to enable persistent sessions:

1. Create a database that can be used for session management. The session manager performs best when it has exclusive use of the database and its connections. It is possible to use an existing database, but it is best to create a new one. For this example we are using DB2.

Open a DB2 command window and type:

```
db2 create db sessdb
```

2. If you are using multiple nodes and a multi-member, clustered environment, then this database needs to be catalogued at each physical machine.
  - a. First, we need to catalog the DB2 node that contains the session database.

If the node is not already cataloged, open a command window and type:

```
db2 catalog tcpip node <node_name> remote <remote_hostname> server  
<service_name>
```

Where:

- <node\_name> is an arbitrary name for identification, for example sessnode.
- <remote\_hostname> is the host name that DB2 can use to connect to the DB2 server containing the session database, for example sesshost.
- <service\_name> is the port number that DB2 is listening to on the remote DB2 server. The default is 50000.

- b. Next, catalog the database at that node using the command:

```
db2 catalog db sessdb as sessdb at node <node_name>
```

Where, <node\_name> is the node name specified in the previous step.

- c. Verify that it is possible to connect to this newly cataloged database. Type the command:

```
db2 connect to sessdbs user <userid> using <password>
```

3. Set up a JDBC provider using the WebSphere administration tools. For information about how to do so, refer to 15.6, “JDBC resources”, of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195.
4. Set up the data source for use with the session manager by using the WebSphere administration tools. Within the JDBC provider that you have just created, configure a new data source. For information about setting up a data source, see 15.7, “Creating a data source”, of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195.

To enable persistent sessions, repeat the following steps for each application server in the cluster:

1. Click **Servers -> Application Servers**.
2. Click **<AppServer Name>**.
3. Click **Web Container -> Session Management -> Distributed Environment Settings**.
4. Select the **Database** radio button and click **Apply**. Click the **Database** link. The window shown in Figure 5-14 on page 177 will be launched.



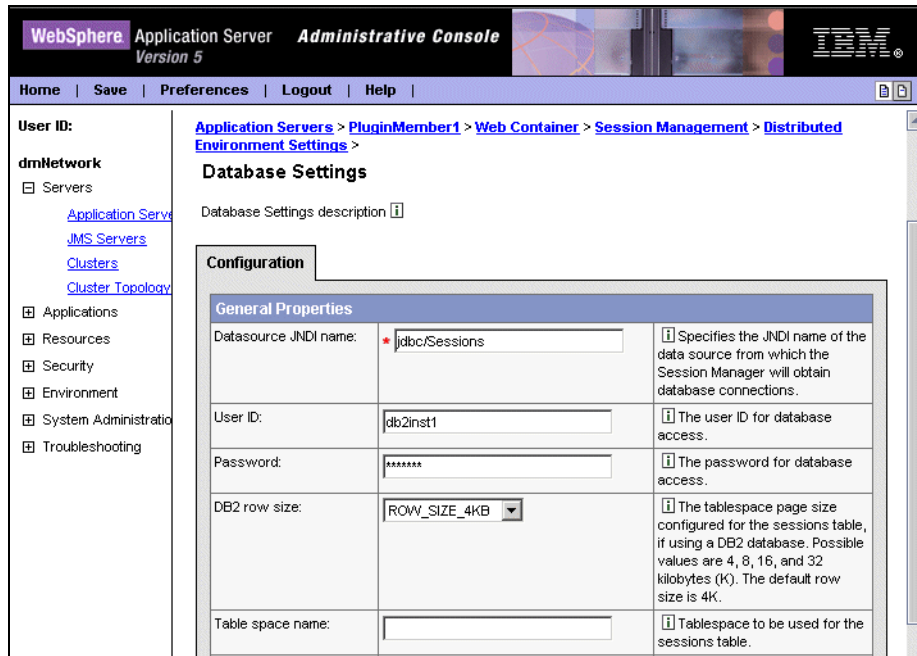


Figure 5-14 Database settings for the session manager

5. Enter values where necessary for the configuration properties:
  - Enter your Datasource JNDI name. The data source must be a non-JTA enabled data source.
  - If required, enter a user ID and password and confirm the password.
  - If you are using DB2 and you anticipate requiring row sizes greater than 4 KB, select the appropriate value from the DB2 row size pull-down. See 14.9.5, “Using larger DB2 page sizes (database persistence)”, of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195 for more information about this setting.
  - If you intend to use a multi-row schema, check the appropriate box. Again, information about multi-row schemes can be found in Chapter 14, “Configuring session management” of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195.
6. Click **Apply**.
7. Repeat these configuration steps for all cluster members.

Upon the first creation of a session, you should see that a table called Session is created for you. The cluster members will now make use of this session database.

In our example we have two cluster members in PluginCluster: PluginMember1 and PluginMember2. If we make an initial session-based request to PluginMember1 and shut it down afterwards, the plug-in will then try to route requests through to PluginMember2. In general, if the plug-in is unable to contact the initial server it will try another cluster member. The next cluster member that responds will check its local cache and, upon not finding the session, will connect to the session database and retrieve the session from there.

When the servlet returns the result to the plug-in, the cluster member that served the page is appended to the list of clone IDs in the session identifier. The session identifiers will now look like those shown in Example 5-10.

*Example 5-10 Example of session JSESSION cookies with multiple clone IDs*

---

JSESSIONID=0002YBY-wJPYdXZvCZkc-LkoBBH:v544d031:v544d0o0

---

The plug-in now tries both of these servers before resorting to the workload management algorithm again. Once the original server is available again, all session requests will return to it.

This behavior means that the plug-in performs faster and the user will go to a server that has a local cached version of the session.

If you wish to test whether the failover and persistent sessions are working, follow the steps in 5.7, “Web server plug-in behavior and failover” on page 184.

## **Memory-to-memory replication**

The new feature of memory-to-memory replication enables the sharing of sessions between application servers without using a database. There are two ways this function can be configured: client-server or peer-peer. In our example we are going to use peer-peer (which is also the default) to depict how session failover works.

For an in-depth explanation of this function, consult Chapter 14, “Configuring session management” of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195.

The benefits of using in-memory replication rather than a database for session persistence is that the overhead and cost of setting up and maintaining a real-time, production database, such as preparing a machine, installing and configuring a database, starting and so on, is not needed. Also, the database becomes a SPOF (Single Point Of Failure) for session persistence and certain cost and effort is required to solve this issue at the database level. However, it was found that database persistence normally performs better than

memory-to-memory replication. See “HTTP sessions and the session management facility” on page 22 for additional information.

All features available in database persistence are available in memory-to-memory replication as well, except for DB2 variable row size and multi-row features, which are features specific to a database.

To effectively deploy memory-to-memory replication in clustered environments, especially large ones, implementers must think carefully how to exactly configure the replicator topology and settings. If care isn’t taken then the amount of memory and resource usage taken by session replication can increase significantly.

The following is a high-level description of how to set up this function. Refer to 7.5, “Configuring WebSphere clusters” on page 276 or, again, to Chapter 14 of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195 for details on how to configure this.

1. Create a replication domain (if not already done so during cluster creation).

Replicator domains define the set of replicator processes that communicate with each other.

2. Add replicator entries.

Replicator entries, or simply replicators, run inside an application server process and define its basic settings, such as a communication port among replicators and a communication port with clients.

3. Configure cluster members (here: PluginMember1 and PluginMember2).

4. Restart the cluster.

In our example we have two cluster members in PluginCluster: PluginMember1 and PluginMember2. If we make an initial session-based request to PluginMember1 and shut it down afterwards, the plug-in will then try and route requests through to PluginMember2. In general, if the plug-in is unable to contact the initial server it will try another cluster member. The next cluster member that responds will already have the session in memory since it has been replicated using memory-to-memory replication.

Again, when the servlet returns the result to the plug-in, the cluster member that served the page is appended to the list of clone IDs in the session identifier. The session identifiers now look like those shown in Example 5-11.

*Example 5-11 Example of session JSESSION cookies with multiple clone IDs*

---

JSESSIONID=0002YBY-wJPYdXZvCZkc-LkoBBH:v544d031:v544d0o0

---

The plug-in now tries both of these servers before resorting to the workload management algorithm again. Once the original server is available again, all session requests will return to it.

If you wish to test whether the failover and persistent sessions are working, follow the steps in 5.7, “Web server plug-in behavior and failover” on page 184.

## 5.6 Troubleshooting the Web server plug-in

If problems occur with the plug-in, the following sections should help in finding a solution.

### 5.6.1 Logging

Logging for the plug-in can occur in three different places:

1. The log specified in the plugin-cfg.xml file (the default is http\_plugin.log).
2. WebSphere Application Server's activity.log.
3. Web server error log.

#### The plug-in log

This log is specified in the plugin-cfg.xml file on the Log element as follows:

```
<Log LogLevel="Trace"
    Name="C:\WebSphere\AppServer\logs\http_plugin.log"/>
```

The LogLevel can be set to:

- ▶ Error
- ▶ Warn
- ▶ Stats (see “LogLevel Stats (new in WebSphere V5.1)” on page 181)
- ▶ Trace (see “Trace” on page 183)

Error contains the least information and Trace contains the most.

If you select **Trace**, then there is much information created for each access to the Web server. This setting should not be used in a normally functioning environment, because the file will rapidly grow and there will be a significant reduction in performance. To learn more about Trace, see 5.6.2, “Trace” on page 183.

The name of the log file is, by default, <WAS\_HOME>/logs/http\_plugin.log. On a remote Web server where WebSphere Application Server is not running, the log file resides on the Web server, not the application server. Each Web server in the environment that is configured with the WebSphere plug-in will write its own copy of the plug-in log to its local file system.

If you wish to change the location of the log, change the name value and a new log will be created and used the next time the plug-in refreshes its configuration; see 5.4.3, “Generation of the plug-in configuration file” on page 151.

Every time the Web server starts, it will create the plug-in log if it does not exist. This is a simple way to see if the plug-in is running. If there is no log created when the Web server is started, then either the plug-in was not initialized or it was unable to create or open the log. Make sure that wherever you place the log, the Web server has the authority to create and modify files.

If there is a problem with settings, creation, or modification of the log, an error will appear in the Web server log.

Successful startup will be signified by lines similar to those shown in Example 5-12 appearing in the log.

*Example 5-12 Message shown on a successful startup on Windows*

---

```
.  
.   
PLUGIN: Plugins loaded.  
PLUGIN: -----System Information-----  
PLUGIN: Bld version: 5.1.0  
PLUGIN: Bld date: Nov  4 2003, 22:35:21  
PLUGIN: Webserver: IBM_HTTP_Server/2.0.42.2 Apache/2.0.46 (Win32)  
PLUGIN: Hostname = web2  
PLUGIN: OS version 5.0, build 2195, 'Service Pack 4'  
PLUGIN: -----  
.   
.
```

---

***LogLevel Stats (new in WebSphere V5.1)***

The plug-in has been enhanced to support a new log level called *Stats*. When this log level is selected, it will list all ERROR and WARNING messages as well as additional server usage statistics and server status used for server selection. For every request the two messages shown in Example 5-13 are logged.

*Example 5-13 Messages for Stats LogLevel*

---

```
[Mon Jul 28 11:30:34 2003] 0004cd88 00000001 - STATS: ws_server_group:  
serverGroupCheckServerStatus: Checking status of Clone271, ignoreWeights 0,  
markedDown 0, retryNow 0, wlbAllows 1 reachedMaxConnectionsLimit 0
```

```
[Mon Jul 28 11:30:34 2003] 0004cd88 00000001 - STATS: ws_server:  
serverSetFailoverStatus: Server Clone271 : pendingConnections 0  
failedConnections 0 affinityConnections 0 totalConnections 1.
```

---

There will be more than one occurrence of the `serverGroupCheckServerStatus` message if the server was not selectable for the current request.

The following server status attributes are used to decide whether the server can handle the request:

- ▶ **ignoreWeights:** This indicates whether the current request should ignore weights. This will be set to 1 if the request has an associated affinity server or if the load balancing is chosen to be random.
- ▶ **markedDown:** This indicates whether the server is currently marked down and can't be selected for the current request.
- ▶ **retryNow:** This indicates that the marked down server can be retried now.
- ▶ **wlbAllows:** This indicates whether the current server has positive weights. A server can be selected only if it has positive weights when weights can't be ignored (`ignoreWeights` set to 0).
- ▶ **reachedMaxConnectionsLimit:** This indicates whether the current server has reached maximum pending connections. A server won't be selected if this attribute has a value of 1.

The following are the additional server statistics. This information is logged after getting the response from the server for the current request. Note that this statistic is collected per process.

- ▶ **pendingConnections:** Number of requests for which response is yet to be received from the server.
- ▶ **failedConnections:** Number of failed requests to the server.
- ▶ **affinityConnections:** Number of requests that are going to an affinity server.
- ▶ **totalConnections:** Total number of requests that are handled by the server.

## WebSphere Application Servers activity.log

The `activity.log` will contain errors that occur when:

1. Generation of the plug-in is launched automatically or manually from the Administrative Console and an error occurs.

When launching the generation of the plug-in and something goes wrong, the errors will be listed in the `activity.log` file. The exception to this is if it is run from the command line. In this case, all errors are written to the screen.

2. An error occurs when the Web container attempts to process a request passed to it by the plug-in. For example, where the information that is in the plugin-cfg.xml does not map to the information in the configuration repository.

If the plug-in configuration is changed manually, it can lead to inconsistencies with the administrative repository, for example if you added a virtual host to the definitions in the plugin-cfg.xml file but did not add the same virtual host to the configuration repository. The error message would be:

```
PLGN0021E: Servlet Request Processor Exception: Virtual Host/WebGroup  
Not Found: The host hostname has not been defined.
```

### **Web server error log**

Errors occur in the error log of the Web server when the plug-in has been unable to access the plug-in log. With IBM HTTP Server, the log is error\_log on AIX and error.log on Windows. It is in the logs directory under the HTTP Server installation directory.

A similar error will occur if the native.log file is locked or inaccessible to the Web server. By default, all users have read, write, and executable permissions for the WebSphere logs directory. This means that there should be no problems with access unless the log location is changed from the default.

## **5.6.2 Trace**

There are two areas of the plug-in configuration where tracing can be enabled: the actions of the plug-in itself, and the generation of the configuration file.

### **Tracing the plug-in**

Tracing is enabled in the plug-in by setting LogLevel to Trace, as specified in 5.6.1, “Logging” on page 180.

A good method for starting a new trace is to change the LogLevel to Trace and change the name of the file to a new value, for example traceplugin.log. This way, you know where your trace starts and you have an easily managed trace file.

Once you are finished, you can return the plug-in configuration file to its original settings and do not need to stop the Web server or the application server for any of this to work.

The trace itself is quite straightforward to follow. If you are tracing requests, you will see a request handled as shown in Figure 5-7 on page 145.

Tracing will provide you with information about:

- ▶ The virtual hosts the plug-in is using. You can view how the virtual host is matched to the request. One of the most common problems is specifying the same alias in multiple virtual hosts and so not giving the plug-in a unique route to follow.
- ▶ The URIs the plug-in is searching. If you are unsure as to why a request is not reaching your application, use a trace to see how the plug-in is matching your browser request.
- ▶ Workload management selection process. Tracing is useful if you wish to validate how your cluster members are being selected and watch failover mechanisms operate.
- ▶ Observing the transport mechanism being used and connections or failures in connection.
- ▶ Session management.

### Tracing GenPluginCfg

It is possible to trace the **GenPluginCfg** command. This will show you how all the information is being gathered to create the configuration file.

To use the trace option, specify the command as:

```
GenPluginCfg -debug yes
```

Refer to Appendix A-17, “GenPluginCfg”, of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195 for more information about this command-line tool.

## 5.7 Web server plug-in behavior and failover

There are various scenarios that can occur in the event of a failure somewhere in your system. This section will examine some example scenarios and show the system behavior.

The tests are designed to access a cluster from a browser via the plug-in in a Web server. The number of nodes, cluster members, and physical servers is specified in the two areas of investigation: Normal operation and Failover operation.

Each of the test setup procedures assumes that the sample application (DefaultApplication.ear) and the BeenThere application has already been installed and that there has been a successful connection to:

- ▶ <http://yourWebserver/snoop>



- ▶ `http://yourWebserver/hitcount`
- ▶ `http://yourWebserver/wlm/beenthere`

Refer to 7.6, “Installing and configuring BeenThere” on page 290 for information about how to obtain, install, and configure BeenThere.

It is also assumed that a persistent session database or a replication domain with replicators has been configured to handle requests for the nodes. See 5.5.4, “Session persistence and failover” on page 173 for details.

### 5.7.1 Normal operation

In this section, we discuss using the WebSphere samples to show how the plug-in achieves workload management and session affinity. These tests were performed in our lab, using the sample topology described in Chapter 7, “Implementing the sample topology” on page 255.

We are interested in how the Web server plug-in distributes browser requests to the available WebSphere Web containers. We stopped our first Web server, web1, to force all browser requests to web2. This way, we only need to observe the plug-in behavior on web2.

#### Workload management with the plug-in

To check the behavior of plug-in workload management:

1. Use the default settings in the plugin-cfg.xml file.
2. Open a browser and go to the URL:

`http://<yourWebServer>/wlm/beenthere`

where `<yourWebServer>` is the host name of your Web server.

3. Click the **Execute** button and observe the Servlet Server Name field.

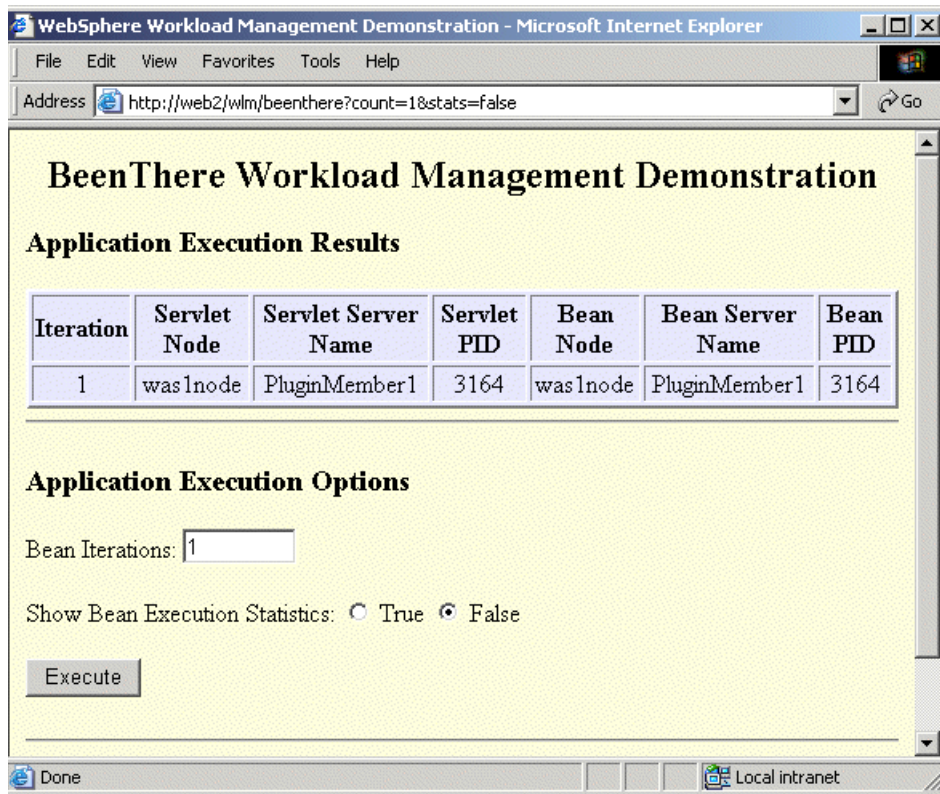


Figure 5-15 BeenThere first click of Execute

- Click the **Execute** button as many times as you have cluster members, and you should see from the Servlet Server Name that each request (clicking **Execute**) is routed to a different cluster member. All cluster members will be cycled through as long as their weights are not set to 0.

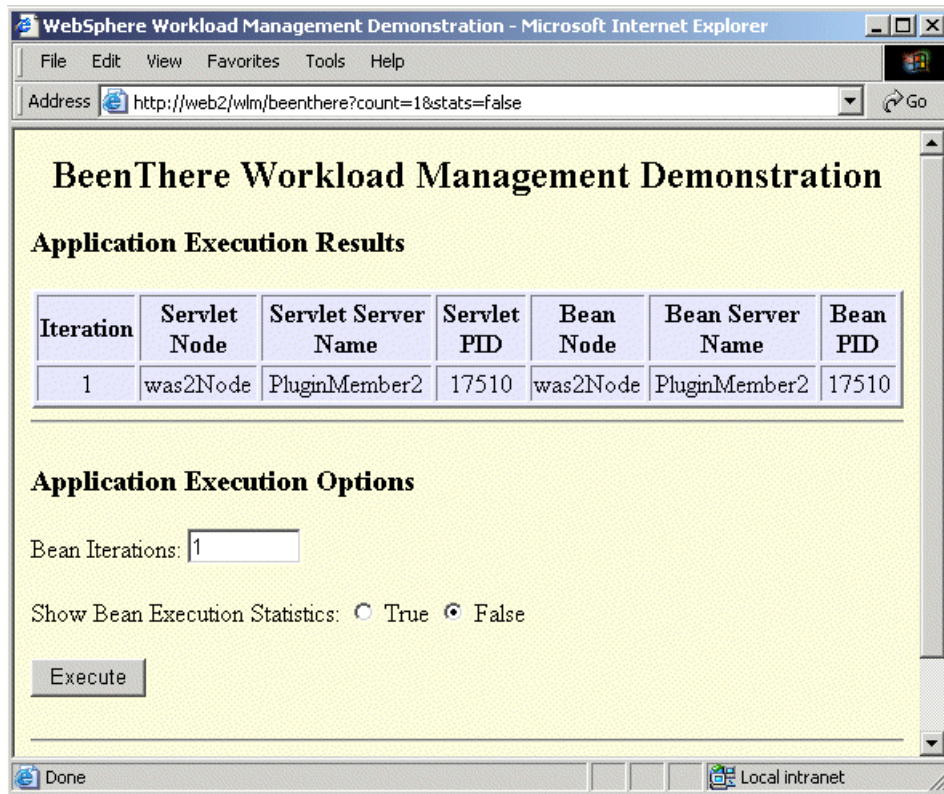


Figure 5-16 BeenThere subsequent click of Execute

**Note:** Changing the bean iteration will not round robin the servlet request. There is only one Web request and multiple EJB requests.

5. Open the plugin-cfg.xml file for editing.
6. Find the line that defines your cluster, the ServerCluster tag containing all your server definitions.
7. Change the line from:

```
<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="PluginCluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
```

to:

```
<ServerCluster CloneSeparatorChange="false" LoadBalance="Random"
Name="PluginCluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
```

8. Save and exit the plugin-cfg.xml file.
9. Restart the Web server or wait for the plug-in to reload the configuration file. Go back to the browser and click **Execute** again.
10. The requests will now be served in a random order, which could mean that multiple clicks of the **Execute** button go to the same cluster member.

## Testing session management

As described before, there are three ways of tracking session management:

- ▶ Cookies
- ▶ SSL ID
- ▶ URL rewriting

### ***Cookies***

We tested session affinity using cookies as follows:

1. In the plugin-cfg.xml file, change the Log tag. Set the Log tag so that the line looks similar to:  

```
<Log LogLevel="Trace" Name="C:\WebSphere\AppServer\logs\sesstrace.log"/>
```
2. Save and close the plugin-cfg.xml file.
3. In the Administrative Console, click **Servers -> Application Servers**.
4. Click **<AppServer Name>**. In this example, PluginMember1 is selected.
5. Click **Web Container -> Session Management**. The window shown in Figure 5-17 on page 189 will appear.
6. Check the **Enable Cookies** check box.
7. Synchronize the nodes and then repeat step 1 to 4 for the other application servers in the cluster. In our case, also PluginMember2 has to be updated.

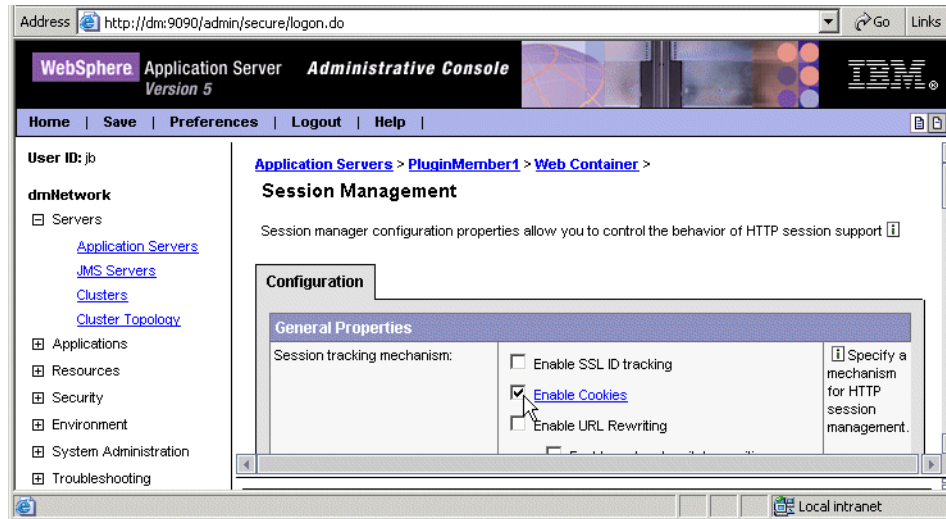


Figure 5-17 Setting Cookies for session management.

8. Stop and start the cluster to make the session manager changes operational.
9. Open a browser and go to:  
`http://<yourWebServer>/hitcount`
10. Select **Session state** from the Generate hit count using options, as shown in Figure 5-18 on page 190.

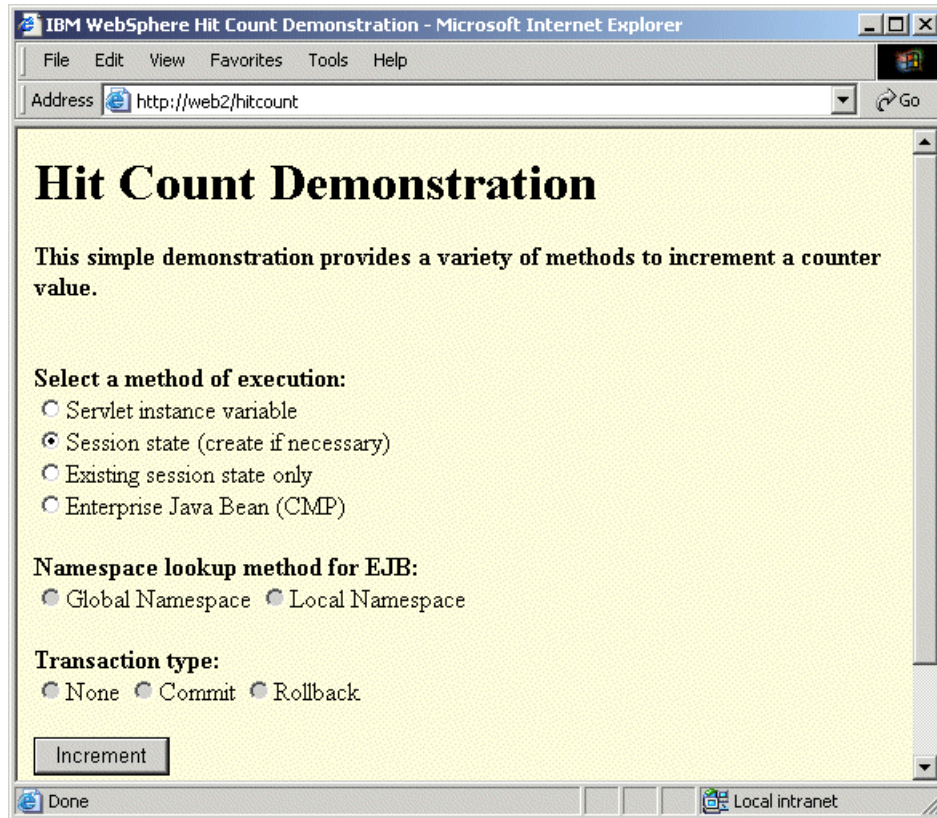


Figure 5-18 HitCount servlet

11. Click the **Increment** button a few times.
12. Open your plug-in log as specified in step 1 on page 188. Look through the processing of the requests and search for `websphereHandleSessionAffinity`. You should see something similar to that shown in Example 5-14 on page 191.

**Note:** If the plug-in log does not contain any trace information, the refresh interval for the plug-in may not have been reached. Wait longer or restart the Web server.

13. Change the Log tag back to its original setting.

```
...
ws_common: websphereWriteRequestReadResponse: Enter
ws_common: websphereHandleSessionAffinity: Checking for session affinity
ws_common: websphereHandleSessionAffinity: Checking the SSL session id
lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'SSLJSESSION'
lib_htrequest: htrequestGetCookieValue: No cookie found for: 'SSLJSESSION'
ws_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID
lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
lib_htrequest: htrequestGetCookieValue: name='JSESSIONID',
value='0000UHHjdWuWJIed1Ziom5Vb6hM:v544d031'
ws_common: websphereParseCloneID: Parsing clone ids from '0000UHHjdWuWJIed1Ziom5Vb6hM:v544d031'
ws_common: websphereParseCloneID: Adding clone id 'v544d031'
ws_common: websphereParseCloneID: Returning list of clone ids
ws_server_group: serverGroupFindClone: Looking for clone
ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
ws_server_group: serverGroupFindClone: Comparing curCloneID 'v544d031' to server clone id
'v544d031'
ws_server_group: serverGroupFindClone: Match for clone 'waslnode_PluginMember1'
ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0, maxConnectionsCount -1.
ws_server_group: serverGroupCheckServerStatus: Checking status of waslnode_PluginMember1,
ignoreWeights 1, markedDown 0, retryNow 0, wlbAllows 2 reachedMaxConnectionsLimit 0
ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0, maxConnectionsCount -1.
ws_server_group: serverGroupIncrementConnectionCount: Server waslnode_PluginMember1 picked,
pendingConnectionCount 1 totalConnectionsCount 3.
ws_common: websphereHandleSessionAffinity: Setting server to waslnode_PluginMember1
ws_server_group: assureWeightsValid: group PluginCluster
ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
ws_server_group: weights_need_reset: waslnode_PluginMember1: 4 max, 2 cur.
ws_server_group: lockedServerGroupUseServer: Server waslnode_PluginMember1 picked, weight 1.
ws_common: websphereFindTransport: Finding the transport
ws_common: websphereFindTransport: Setting the transport(case 2): appl.itso.ibm.com on port
9088
ws_common: websphereExecute: Executing the transaction with the app server
ws_common: websphereGetStream: Getting the stream to the app server
ws_transport: transportStreamDequeue: Checking for existing stream from the queue
ws_common: websphereSocketIsClosed: Checking to see if socket is still open
ws_common: websphereGetStream: Using existing stream from transport queue
...
```

---

### **SSL ID**

We tested session affinity using SSL ID and cookies as follows:

1. Set up your Web server for SSL traffic. See the relevant Web server documentation for details on this.

2. In the Administrative Console, go to the virtual host definitions by clicking **Environment -> Virtual Hosts -> PluginVirtualHost -> Host Aliases -> New**.
3. Add the host name as web1 and the port as 443 and click **Apply**.
4. Go to **Servers -> ApplicationServers -> <AppServer Name> -> Web Container -> Session Management**. Select **Enable SSL ID tracking** and **Enable Cookies**. We did this at the application server level so that all other session management levels would be overwritten. However, if you have overwrite session management set at the application or Web module level this will not work.
5. Click **Apply** and synchronize the changes.
6. Repeat steps 4 and 5 for PluginMember2.
7. Stop and start your cluster to make the session manager changes operational.
8. Regenerate the plug-in configuration file for your Web servers.
9. In the plugin-cfg.xml file, change the Log tag. Set the Log tag so that the line looks similar to:
 

```
<Log LogLevel="Trace"
      Name="/usr/WebSphere/AppServer/logs/sesstrace.log"/>
```
10. Save and close the plugin-cfg.xml file.
11. Open a browser and go to:
 

```
https://<yourWebServer>/hitcount
```

Make sure to connect using https from your Web server.
12. From the Select a method of execution option, select **Session state** (see Figure 5-18 on page 190).
13. Click the **Increment** button a few times.
14. Open your plug-in log as specified in step 9. Look through the processing of the requests and search for websphereHandleSessionAffinity.
 

You should see something similar to that shown in Example 5-15 on page 193.
15. Change the Log tag back to its original setting.



**Note:** When using SSL between the client and the Web server, WebSphere Application Server V5.1 will try to use SSL communication between the plug-in and the Web container. To disable this, just delete the HTTPS transport lines for the server in plugin-cfg.xml. Otherwise, refer to the redbook *IBM WebSphere V5.0 Security*, SG24-6573 on details of how to set up HTTPS communication between the plug-in and Web container.

*Example 5-15 Plug-in trace when using SSL ID and cookies*

---

```
...
ws_common: websphereWriteRequestReadResponse: Enter
ws_common: websphereHandleSessionAffinity: Checking for session affinity
ws_common: websphereHandleSessionAffinity: Checking the SSL session id
lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'SSLJSESSION'
lib_htrequest: htrequestGetCookieValue: name='SSLJSESSION',
value='0001SESSIONMANAGEMENTAFFINI:v544d031'
ws_common: websphereParseCloneID: Parsing clone ids from '0001SESSIONMANAGEMENTAFFINI:v544d031'
ws_common: websphereParseCloneID: Adding clone id 'v544d031'
ws_common: websphereParseCloneID: Returning list of clone ids
ws_server_group: serverGroupFindClone: Looking for clone
ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
ws_server_group: serverGroupFindClone: Comparing curCloneID 'v544d031' to server clone id
'v544d031'
ws_server_group: serverGroupFindClone: Match for clone 'waslnode_PluginMember1'
ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0, maxConnectionsCount -1.
ws_server_group: serverGroupCheckServerStatus: Checking status of waslnode_PluginMember1,
ignoreWeights 1, markedDown 0, retryNow 0, wlbAllows 3 reachedMaxConnectionsLimit 0
ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0, maxConnectionsCount -1.
ws_server_group: serverGroupIncrementConnectionCount: Server waslnode_PluginMember1 picked,
pendingConnectionCount 1 totalConnectionsCount 2.
ws_common: websphereHandleSessionAffinity: Setting server to waslnode_PluginMember1
ws_server_group: assureWeightsValid: group PluginCluster
ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
ws_server_group: weights_need_reset: waslnode_PluginMember1: 4 max, 3 cur.
ws_server_group: lockedServerGroupUseServer: Server waslnode_PluginMember1 picked, weight 2.
ws_common: websphereFindTransport: Finding the transport
ws_common: websphereFindTransport: Setting the transport(case 1): appl.itso.ibm.com on port
9447
ws_common: websphereExecute: Executing the transaction with the app server
ws_common: websphereGetStream: Getting the stream to the app server
ws_transport: transportStreamDequeue: Checking for existing stream from the queue
ws_common: websphereSocketIsClosed: Checking to see if socket is still open
ws_common: websphereGetStream: Using existing stream from transport queue
...
```

---

## URL rewriting

There is no servlet provided with the WebSphere samples that uses URL rewriting. To perform this test, you can either use your own example or the sample given in Appendix B, “Sample URL rewrite servlet” on page 929.

We tested session affinity using the URL rewrite sample given in Appendix B, “Sample URL rewrite servlet” on page 929 as follows:

1. Set up the URL rewrite sample application. Make sure you regenerate the plug-in configuration on your Web server.
2. In the plugin-cfg.xml file, change the Log tag. Set the Log tag so that the line looks similar to:

```
<Log LogLevel="Trace" Name="C:\WebSphere\AppServer\logs\sesstrace.log"/>
```

3. Save and close the plugin-cfg.xml file.
4. In the Administrative Console, select **Servers -> Application Servers**.
5. Click **<AppServer Name>**. In this example, PluginMember1 is selected.
6. Click **Web Container -> Session Management**. The window shown in Figure 5-19 will appear.
7. Click the **Enable URL Rewriting** check box.



Figure 5-19 Setting URL rewriting for session management.

8. Synchronize the nodes and then repeat step 4 to 7 for the rest of the cluster members. In our case PluginMember2 has to be updated also.
9. Stop and start your cluster to make the session manager changes operational.

10. Open a browser and go to:

`http://<yourWebServer>/urltest/urltest`

11. Click the **Request this servlet again using the rewritten URL** link, as shown in Figure 5-20.

12. Click the **Request this servlet again using the rewritten URL** link a few more times.

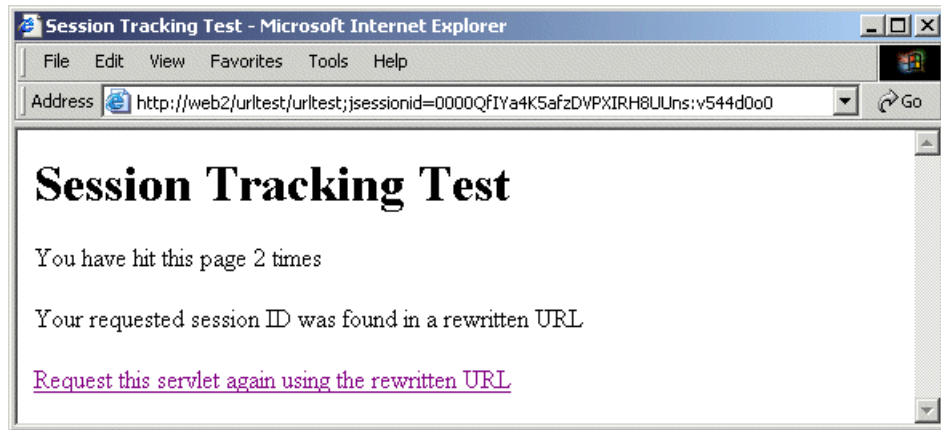


Figure 5-20 URL rewriting example

13. Open your plug-in log as specified in step 2. Look through the processing of the requests and search for `websphereHandleSessionAffinity`.

You should see something similar to that shown in Example 5-16.

14. Change the Log tag back to its original setting.

*Example 5-16 Plug-in trace when using URL rewriting*

```
...
ws_common: websphereWriteRequestReadResponse: Enter
ws_common: websphereHandleSessionAffinity: Checking for session affinity
ws_common: websphereHandleSessionAffinity: Checking the SSL session id
lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'SSLJSESSION'
lib_htrequest: htrequestGetCookieValue: No cookie found for: 'SSLJSESSION'
ws_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID
lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
lib_htrequest: htrequestGetCookieValue: No cookie found for: 'JSESSIONID'
ws_common: websphereHandleSessionAffinity: Checking the url rewrite affinity: jsessionid
ws_common: websphereParseSessionID: Parsing session id from
'/urltest/urltest;jsessionid=0000QfIYa4K5afzDVPXIRH8UUns:v544d0o0'
ws_common: websphereParseSessionID: Parsed session id
'jsessionid=0000QfIYa4K5afzDVPXIRH8UUns:v544d0o0'
```

```

ws_common: websphereParseCloneID: Parsing clone ids from
'/urltest/urltest;jsessionid=0000QfIYa4K5afzDVPXIRH8UUns:v544d0o0'
ws_common: websphereParseCloneID: Adding clone id 'v544d0o0'
ws_common: websphereParseCloneID: Returning list of clone ids
ws_server_group: serverGroupFindClone: Looking for clone
ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
ws_server_group: serverGroupFindClone: Comparing curCloneID 'v544d0o0' to server clone id
'v544d031'
ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
ws_server_group: serverGroupFindClone: Comparing curCloneID 'v544d0o0' to server clone id
'v544d0o0'
ws_server_group: serverGroupFindClone: Match for clone 'was2Node_PluginMember2'
ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0, maxConnectionsCount -1.
ws_server_group: serverGroupCheckServerStatus: Checking status of was2Node_PluginMember2,
ignoreWeights 1, markedDown 0, retryNow 0, wlbAllows 0 reachedMaxConnectionsLimit 0
ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0, maxConnectionsCount -1.
ws_server_group: serverGroupIncrementConnectionCount: Server was2Node_PluginMember2 picked,
pendingConnectionCount 1 totalConnectionsCount 2.
ws_common: websphereHandleSessionAffinity: Setting server to was2Node_PluginMember2
ws_server_group: assureWeightsValid: group PluginCluster
ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
ws_server_group: weights_need_reset: was1Node_PluginMember1: 4 max, 4 cur.
ws_server_group: lockedServerGroupUseServer: Server was2Node_PluginMember2 picked, weight -1.
ws_common: websphereFindTransport: Finding the transport
ws_common: websphereFindTransport: Setting the transport(case 2): app2.itso.ibm.com on port
9086
ws_common: websphereExecute: Executing the transaction with the app server
ws_common: websphereGetStream: Getting the stream to the app server
ws_transport: transportStreamDequeue: Checking for existing stream from the queue
ws_common: websphereSocketIsClosed: Checking to see if socket is still open
ws_common: websphereGetStream: Using existing stream from transport queue
...

```

---

## 5.7.2 Failover operation

This section shows what happens when certain failover situations are forced with the Web container. These tests were performed in our lab, using the sample topology described in Chapter 7, “Implementing the sample topology” on page 255.

We stopped the second Web server, web2, to force all browser requests to web1. This way, we only need to observe the plug-in behavior on web1.

## Stopping a cluster member

We tested plug-in failover operation with a stopped cluster member, as follows:

1. Verify that all the cluster members are running within the cluster. Check this by following the steps described in “Workload management with the plug-in” on page 185.

Cycle through to make sure that all of the cluster members are available. There is no need to repeat the changes to the workload management policy.

2. In the plugin-cfg.xml file, change the Log tag. Set the Log tag so that the line looks like this:

```
<Log LogLevel="Trace"
      Name="C:\WebSphere\AppServer\logs\stopclustermember.log"/>
```

3. Save and close the plugin-cfg.xml file.
4. Open the Administrative Console and select one of your cluster members.
5. Stop this cluster member.
6. Repeat step 1, noting the absence of the cluster member just stopped.
7. Start the cluster member again.
8. Wait 60 seconds and repeat step 1. The cluster member will return to serving requests.

### ***What is happening?***

The plug-in uses the round robin method to distribute the requests to the cluster members. Upon reaching the cluster member that was stopped, the plug-in attempts to connect and finds there is no HTTP process listening on the port.

The plug-in marks this cluster member as down and writes an error to the log, as shown in Example 5-17.

#### *Example 5-17 Plug-in trace with cluster member down*

---

```
...
ERROR: ws_common: websphereGetStream: Failed to connect to app server on host
'app1.itso.ibm.com', OS err=10061
ERROR: ws_common: websphereExecute: Failed to create the stream
ERROR: ws_server: serverSetFailoverStatus: Marking waslnode_PluginMember1 down
STATS: ws_server: serverSetFailoverStatus: Server waslnode_PluginMember1 :
pendingConnections 0 failedConnections 1 affinityConnections 0 totalConnections
0.
ERROR: ws_common: websphereHandleRequest: Failed to execute the transaction to
'waslnode_PluginMember1' on host 'app1.itso.ibm.com'; will try another one
...

```

---

It then tries to connect to the next cluster member in the primary server list. When it has found a cluster member that works, the request is served from that cluster member instead.

The plug-in does not try the cluster member for another 60 seconds. If tracing is enabled, you will be able to see that the plug-in shows the time left every time it comes to the downed cluster member in the round robin algorithm, as shown in Example 5-18.

*Example 5-18 Plug-in trace cluster member retry interval countdown*

---

```
...
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of
waslnode_PluginMember1, ignoreWeights 0, markedDown 1, retryNow 0, wlbAllows 0
reachedMaxConnectionsLimit 0
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
TRACE: ws_server_group: serverGroupCheckServerStatus: Server waslnode_PluginMember1 is marked
down; retry in 55
...
```

---

After restarting the cluster member and once the 60-second retry interval has passed, the next request attempt to the downed cluster member tries to connect again. This time, it is successful and the request is served.

## **Stopping a cluster member containing active sessions**

When using sessions, the impact of stopping or killing a cluster member is that session information in that cluster member is lost. This means that persistent or replicated sessions need to be used to allow for the failover of the session to another cluster member. Follow this procedure to discover what happens:

1. Make sure persistent or replicated sessions have been enabled. See 5.5.4, “Session persistence and failover” on page 173 for details.
2. Set up and test session tracking using cookies, as described in “Cookies” on page 188.
3. Verify that all the cluster members are running within the cluster. Check this by following the steps described in “Workload management with the plug-in” on page 185.

Cycle through to make sure that all the cluster members are available. There is no need to repeat the changes to the workload management policy.

**Note:** If you use the same browser for the session management test and the workload management test, you will find that all your BeenThere requests return to the same server. This is because the cookie for your session management test is still valid.

4. Return to the session tracking test at:

`http://<yourWebServer>/hitcount`

Click the **Increment** button a few times and remember the session count you have reached.

5. Check the log to see which cluster member served your session, referring to Example 5-14 on page 191 to see what to look for. Alternatively:

- a. From the same browser, point to:

`http://<yourWebServer>/wlm/beenthere`

- b. Note the servlet server name, since this is the cluster member that is serving your session.

- c. Return to the session tracking test at:

`http://<yourWebServer>/hitcount`

6. Stop the cluster member that you located in the log file or using BeenThere.
7. Increment the session count in your Web browser. You will see that the session count should continue from the number you noted in step 4.
8. Start the downed cluster member again.
9. Repeat step 7. The session counter will still continue from the previous number and no new session will be created.

### ***What is happening?***

Upon choosing to increment the session counter, WebSphere adds a cookie to your browser. Within this cookie is the session identifier and a clone ID of where this session is stored.

At the end of each request, the session is written to the database or distributed between all application servers using memory-to-memory replication. When the cluster member running the session is stopped, the plug-in chooses another cluster member for the request to go to instead. In the case of using database persistence, this cluster member, finding that it does not have the session cached locally, searches the database. In the case of using memory-to-memory replication, the application server will find the session in its own cache. So in both cases, the other application server finds the session and uses that one instead of creating a new one. This is why the session counter is incremented, not reset.

The WebSphere session manager changes the cookie and appends the cluster clone ID of the new cluster member to it. The cookie now has two cluster clone IDs on it, the original (stopped) cluster member and the failover cluster member.

The plug-in now tries to connect to the stopped cluster member first, and finding it marked as down, will try the failover cluster member instead.

Starting up the stopped cluster member means that the plug-in now returns the session requests to the original cluster member after the retry interval has passed.

This can be seen in the trace listing in Example 5-19.

*Example 5-19 Plug-in trace with session failover*

---

```
...
### We have a session with clone id 'v544d031' for PluginMember1
TRACE: ws_server: serverCreate: Creating the server object
TRACE: ws_server: serverSetName: Setting name was1node_PluginMember1
TRACE: ws_server: serverSetCloneID: Setting clone id v544d031

### We have a session with clone id 'v544d0o0' for PluginMember2
TRACE: ws_server: serverCreate: Creating the server object
TRACE: ws_server: serverSetName: Setting name was2Node_PluginMember2
TRACE: ws_server: serverSetCloneID: Setting clone id v544d0o0
...
### When a request comes, the plugin dispatches it to PluginMember1
TRACE: ws_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
TRACE: lib_htrequest: htrequestGetCookieValue: name='JSESSIONID',
value='0001IWuUT_zhR-gFYB-p0Ak75Q5:v544d031'
TRACE: ws_common: websphereParseCloneID: Parsing clone ids from
'0001IWuUT_zhR-gFYB-p0Ak75Q5:v544d031'
TRACE: ws_common: websphereParseCloneID: Adding clone id 'v544d031'
TRACE: ws_common: websphereParseCloneID: Returning list of clone ids
TRACE: ws_server_group: serverGroupFindClone: Looking for clone
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID 'v544d031' to server clone
id 'v544d031'
TRACE: ws_server_group: serverGroupFindClone: Match for clone 'was1node_PluginMember1'
...
### When PluginMember1 goes down a connection to the next cluster member in the the primary
server list will be attempted. Note that the plugin will first try to connect to PluginMember1,
when it can't connect to it, it will restart the process of checking for cluster members
starting at the first cluster member in the primary server list.
.
TRACE: ws_transport: transportStreamDequeue: Checking for existing stream from the queue
```



```

ERROR: ws_common: websphereGetStream: Failed to connect to app server on host
'appl.itso.ibm.com', OS err=10061
ERROR: ws_common: websphereExecute: Failed to create the stream
ERROR: ws_server: serverSetFailoverStatus: Marking waslnode_PluginMember1 down
STATS: ws_server: serverSetFailoverStatus: Server waslnode_PluginMember1 : pendingConnections 0
failedConnections 1 affinityConnections 2 totalConnections 2.
ERROR: ws_common: websphereHandleRequest: Failed to execute the transaction to
'waslnode_PluginMember1' on host 'appl.itso.ibm.com'; will try another one
...
### Then it iterates again descending down the primary server list.
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
TRACE: lib_htrequest: htrequestGetCookieValue: name='JSESSIONID',
value='0001IWuUT_zhR-gFYB-p0Ak75Q5:v544d031'
TRACE: ws_common: websphereParseCloneID: Parsing clone ids from
'0001IWuUT_zhR-gFYB-p0Ak75Q5:v544d031'
TRACE: ws_common: websphereParseCloneID: Adding clone id 'v544d031'
TRACE: ws_common: websphereParseCloneID: Returning list of clone ids
TRACE: ws_server_group: serverGroupFindClone: Looking for clone
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID 'v544d031' to server clone
id 'v544d031'
TRACE: ws_server_group: serverGroupFindClone: Match for clone 'waslnode_PluginMember1'
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of
waslnode_PluginMember1, ignoreWeights 1, markedDown 1, retryNow 0, wlbAllows 0
reachedMaxConnectionsLimit 0
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
TRACE: ws_server_group: serverGroupCheckServerStatus: Server waslnode_PluginMember1 is marked
down; retry in 60
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID 'v544d031' to server clone
id 'v544d000'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupGetNextUpPrimaryServer: Getting the next up primary server
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of
was2Node_PluginMember2, ignoreWeights 1, markedDown 0, retryNow 0, wlbAllows 0
reachedMaxConnectionsLimit 0
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
TRACE: ws_server_group: serverGroupIncrementConnectionCount: Server was2Node_PluginMember2
picked, pendingConnectionCount 1 totalConnectionsCount 2.
TRACE: ws_server_group: serverGroupFindClone: Returning next up server was2Node_PluginMember2
TRACE: ws_common: websphereHandleSessionAffinity: Setting server to was2Node_PluginMember2
...

```

```

### On the the response back from PluginMember2 a new clone id is appended to the session
cookie
TRACE: ws_common: websphereExecute: Wrote the request; reading the response
TRACE: lib_htresponse: htresponseRead: Reading the response: b2c384
TRACE: HTTP/1.1 200 OK
TRACE: Server: WebSphere Application Server/5.1
TRACE: Cache-Control: no-cache
TRACE: Set-Cookie: JSESSIONID=0002IWuUT_zhR-gFYB-p0Ak75Q5:v544d031:v544d0o0;Path=/
...
### On the the next request we now have a second clone id on the session cookie.
TRACE: ws_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
TRACE: lib_htrequest: htrequestGetCookieValue: name='JSESSIONID',
value='0002IWuUT_zhR-gFYB-p0Ak75Q5:v544d031:v544d0o0'
TRACE: ws_common: websphereParseCloneID: Parsing clone ids from
'0002IWuUT_zhR-gFYB-p0Ak75Q5:v544d031:v544d0o0'
TRACE: ws_common: websphereParseCloneID: Adding clone id 'v544d031'
TRACE: ws_common: websphereParseCloneID: Adding clone id 'v544d0o0'
TRACE: ws_common: websphereParseCloneID: Returning list of clone ids
TRACE: ws_server_group: serverGroupFindClone: Looking for clone
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID 'v544d031' to server clone
id 'v544d031'
TRACE: ws_server_group: serverGroupFindClone: Match for clone 'waslnode_PluginMember1'
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of
waslnode_PluginMember1, ignoreWeights 1, markedDown 1, retryNow 0, wlbAllows 0
reachedMaxConnectionsLimit 0
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
TRACE: ws_server_group: serverGroupCheckServerStatus: Server waslnode_PluginMember1 is marked
down; retry in 55
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID 'v544d031' to server clone
id 'v544d0o0'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID 'v544d0o0' to server clone
id 'v544d031'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID 'v544d0o0' to server clone
id 'v544d0o0'
TRACE: ws_server_group: serverGroupFindClone: Match for clone 'was2Node_PluginMember2'
...
### When PluginMember1 comes back up after the retry interval our session requests go back to
it

```

TRACE: ws\_common: websphereHandleSessionAffinity: Checking the cookie affinity: JSESSIONID  
TRACE: lib\_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'  
TRACE: lib\_htrequest: htrequestGetCookieValue: name='JSESSIONID',  
value='0002IWuUT\_zhR-gFYB-p0Ak75Q5:v544d031:v544d0o0'  
TRACE: ws\_common: websphereParseCloneID: Parsing clone ids from  
'0002IWuUT\_zhR-gFYB-p0Ak75Q5:v544d031:v544d0o0'  
TRACE: ws\_common: websphereParseCloneID: Adding clone id 'v544d031'  
TRACE: ws\_common: websphereParseCloneID: Adding clone id 'v544d0o0'  
TRACE: ws\_common: websphereParseCloneID: Returning list of clone ids  
TRACE: ws\_server\_group: serverGroupFindClone: Looking for clone  
TRACE: ws\_server\_group: serverGroupGetFirstPrimaryServer: getting the first primary server  
TRACE: ws\_server\_group: serverGroupFindClone: Comparing curCloneID 'v544d031' to server clone  
id 'v544d031'  
TRACE: ws\_server\_group: serverGroupFindClone: Match for clone 'was1node\_PluginMember1'  
TRACE: ws\_server: serverHasReachedMaxConnections: currentConnectionsCount 0,  
maxConnectionsCount -1.  
STATS: ws\_server\_group: serverGroupCheckServerStatus: Checking status of  
was1node\_PluginMember1, ignoreWeights 1, markedDown 1, retryNow 1, wlbAllows 0  
reachedMaxConnectionsLimit 0  
TRACE: ws\_server: serverHasReachedMaxConnections: currentConnectionsCount 0,  
maxConnectionsCount -1.  
TRACE: ws\_server\_group: serverGroupCheckServerStatus: Time to retry server  
was1node\_PluginMember1  
TRACE: ws\_server\_group: serverGroupIncrementConnectionCount: Server was1node\_PluginMember1  
picked, pendingConnectionCount 1 totalConnectionsCount 3.  
TRACE: ws\_common: websphereHandleSessionAffinity: Setting server to was1node\_PluginMember1  
TRACE: ws\_server\_group: assureWeightsValid: group PluginCluster  
TRACE: ws\_server\_group: serverGroupGetFirstPrimaryServer: getting the first primary server  
TRACE: ws\_server\_group: weights\_need\_reset: was1node\_PluginMember1: 4 max, 0 cur.  
TRACE: ws\_server\_group: serverGroupGetNextPrimaryServer: getting the next primary server  
TRACE: ws\_server\_group: weights\_need\_reset: was2Node\_PluginMember2: 1 max, 0 cur.  
TRACE: ws\_server\_group: serverGroupGetNextPrimaryServer: getting the next primary server  
TRACE: ws\_server\_group: weights\_need\_reset: Time to reset the weights  
TRACE: ws\_server\_group: serverGroupGetFirstServer: getting the first server  
TRACE: ws\_server\_group: serverGroupGetNextServer: getting the next server  
TRACE: ws\_server\_group: serverGroupGetNextServer: getting the next server  
TRACE: ws\_server\_group: assureWeightsValid: max multiplication factor 1.  
TRACE: ws\_server\_group: serverGroupGetFirstServer: getting the first server  
TRACE: ws\_server\_group: assureWeightsValid: Server was2Node\_PluginMember2: 1 max, 1 cur.  
TRACE: ws\_server\_group: serverGroupGetNextServer: getting the next server  
TRACE: ws\_server\_group: serverGroupGetNextServer: getting the next server  
TRACE: ws\_server\_group: lockedServerGroupUseServer: Server was1node\_PluginMember1 picked,  
weight -1.  
TRACE: ws\_common: websphereFindTransport: Finding the transport  
...

---

## Killing a cluster member during a request

Killing the Java virtual machine (JVM) that a cluster member is running on is the best way to see what will happen if a cluster member crashes *during* a request. When killing a cluster member *before* a request, the effect is the same as seen in “Stopping a cluster member” on page 197.

When you kill a cluster member during a request, sometimes that request is dispatched to another cluster member. There is a mechanism for the plug-in to swap over the request in the middle of the process. However, this mechanism does not guarantee high-availability of all requests. In such case, a cluster member that is functioning will be chosen for a re-issued request, which can then be tried again. The request will then be completed.

If you wish to see this, follow these steps.

1. Verify that all the cluster members are running within the cluster. Check this by following the steps described in “Workload management with the plug-in” on page 185.

Cycle through to make sure that all the cluster members are available. There is no need to repeat the changes to the workload management policy.

2. In the plugin-cfg.xml file, change the Log tag. Set the Log tag so that the line looks like this:

```
<Log LogLevel="Trace"
      Name="C:\WebSphere\AppServer\logs\killclustermember.log"/>
```

3. Save and close the plugin-cfg.xml file.
4. Find the Java process ID of the application server cluster member you want to kill. On AIX, we used the following command to find the process ID of our cluster member named PluginMember2:

```
ps -ef | grep PluginMember2
```

Note the process ID of your cluster member.

5. Open a browser and go to:

```
http://<yourWebServer>/wlm/beenthere
```

6. Click the **Execute** button until the Servlet Server Name field shows the name of the cluster member *before* the one you are going to kill. Monitor how the round robin process is cycling to find this out. The next time that you click **Execute**, the request needs to go to the cluster member that you intend to kill. This assumes that you are using application servers with equal weights or you need to make sure that none of your application servers reaches a weight of 0 while performing this test.
7. Set the Bean Iterations field to a large amount, for example 10000.

8. If you are running on AIX, now is a good time to use the **tail** command to monitor your plug-in log file.
9. Click the **Execute** button. This will start the request running on your cluster member. If you look at the plug-in log, you see that there is information passing between browser and application server.
10. Kill the cluster member you located in step 4 on page 204. The request will subsequently be rerouted to another cluster member. In this case in Example 5-20, the request is dispatched to another cluster member.

*Example 5-20 Plug-in trace with request failover*

---

```
...
### At first, the request is processing by the PluginMember2
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
TRACE: ws_server_group: lockedServerGroupUseServer: Server
was2Node_PluginMember2 picked, weight 0.
TRACE: ws_server_group: serverGroupIncrementConnectionCount: Server
was2Node_PluginMember2 picked, pendingConnectionCount 1 totalConnectionsCount
9.
TRACE: ws_common: websphereFindTransport: Finding the transport
...
### The plugin detects the error of the PluginMember2 and the request is
dispatched to the PluginMember1 again
ERROR: ws_common: websphereExecute: Failed to read from a new stream; App
Server may have gone down during read
TRACE: lib_stream: destroyStream: Destroying the stream
STATS: ws_server: serverSetFailoverStatus: Server was2Node_PluginMember2 :
pendingConnections 0 failedConnections 2 affinityConnections 0 totalConnections
8.
ERROR: ws_common: websphereHandleRequest: Failed to execute the transaction to
'was2Node_PluginMember2' on host 'app2.itso.ibm.com'; will try another one
TRACE: ws_common: websphereHandleSessionAffinity: Checking for session affinity
TRACE: ws_common: websphereHandleSessionAffinity: Checking the SSL session id
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie:
'SSLJSESSION'
TRACE: lib_htrequest: htrequestGetCookieValue: No cookie found for:
'SSLJSESSION'
TRACE: ws_common: websphereHandleSessionAffinity: Checking the cookie affinity:
JSESSIONID
TRACE: lib_htrequest: htrequestGetCookieValue: Looking for cookie: 'JSESSIONID'
TRACE: lib_htrequest: htrequestGetCookieValue: No cookie found for:
'JSESSIONID'
TRACE: ws_common: websphereHandleSessionAffinity: Checking the url rewrite
affinity: jsessionid
TRACE: ws_common: websphereParseSessionID: Parsing session id from
'/wlm/beenthere'
TRACE: ws_common: websphereParseSessionID: Failed to parse session id
```

```

TRACE: ws_server_group: serverGroupNextRoundRobinServer: Round Robin load
balancing
TRACE: ws_server_group: serverGroupNextRoundRobinServer: numPrimaryServers is 2
TRACE: ws_server_group: assureWeightsValid: group PluginCluster
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first
primary server
TRACE: ws_server_group: weights_need_reset: was1node_PluginMember1: 1 max, 0
cur.
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next
primary server
TRACE: ws_server_group: weights_need_reset: was2Node_PluginMember2: 1 max, 0
cur.
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next
primary server
TRACE: ws_server_group: weights_need_reset: Time to reset the weights
TRACE: ws_server_group: serverGroupGetFirstServer: getting the first server
TRACE: ws_server_group: serverGroupGetNextServer: getting the next server
TRACE: ws_server_group: serverGroupGetNextServer: getting the next server
TRACE: ws_server_group: assureWeightsValid: max multiplication factor 1.
TRACE: ws_server_group: serverGroupGetFirstServer: getting the first server
TRACE: ws_server_group: assureWeightsValid: Server was2Node_PluginMember2: 1
max, 1 cur.
TRACE: ws_server_group: serverGroupGetNextServer: getting the next server
TRACE: ws_server_group: assureWeightsValid: Server was1node_PluginMember1: 1
max, 1 cur.
TRACE: ws_server_group: serverGroupGetNextServer: getting the next server
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next
primary server
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first
primary server
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of
was1node_PluginMember1, ignoreWeights 0, markedDown 0, retryNow 0, wlbAllows 1
reachedMaxConnectionsLimit 0
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount -1.
TRACE: ws_server_group: lockedServerGroupUseServer: Server
was1node_PluginMember1 picked, weight 0.
TRACE: ws_server_group: serverGroupIncrementConnectionCount: Server
was1node_PluginMember1 picked, pendingConnectionCount 1 totalConnectionsCount
26.
TRACE: ws_common: websphereFindTransport: Finding the transport
TRACE: ws_common: websphereFindTransport: Setting the transport(case 2):
appl.itso.ibm.com on port 9088

```

...

---

## Overloading a cluster member

There are only so many concurrent requests that can be handled by a Web container in a cluster member. The number of concurrent requests is determined by the maximum number of threads available (10 threads implies 10 concurrent requests). However, a request does not necessarily constitute a user request. A browser might make multiple requests to get the information a user requested.

Connections coming into the Web container's embedded HTTP transport feed requests to threads. If there are more connections than threads available, connections start to backlog, waiting for free threads. There is a maximum setting for the number of connections that can be backlogged as well. If the maximum number of connections in the backlog is exceeded, then no more connections will be allowed to the embedded HTTP server's port.

If there has been a successful connection but it is waiting for a thread in the Web container, then the plug-in will wait for a response (and so will the client). If the connection backlog is full, the plug-in will be refused a connection to the port and the plug-in will treat this in the same way as a stopped cluster member. It will mark the cluster member as down and select a new cluster member to which to pass the request. The cluster member will then hopefully be able to reduce its connection backlog, since the plug-in will not try it again until the retry interval passes.

To avoid this overloading of cluster members, your environment needs to be configured to accept the load you are expecting. This includes the setting of weights that correspond to the system capabilities, the correct balance of cluster members and Web servers, and setting up the queues for requests and connections.

To monitor the behavior of the plug-in when a cluster member has too many requests, use a load testing tool (such as ApacheBench or JMeter), plug-in trace, and Tivoli Performance Viewer.

### 5.7.3 Tuning failover

There are a few places where failover can be tuned. This section details these settings.

#### ConnectTimeout setting

When a cluster member exists on a machine that is removed from the network (because its network cable is unplugged or it has been powered off, for example), the plug-in, by default, cannot determine the cluster member's status until the operating system TCP/IP timeout expires. Only then will the plug-in be able to forward the request to another available cluster member.

It is not possible to change the operating system timeout value without side effects. For instance, it might make sense to change this value to a low setting so that the plug-in can fail over quickly.

However, the timeout value on some of the operating systems is not only used for outgoing traffic (from Web server to application server) but also for incoming traffic. This means that any changes to this value will also change the time it takes for clients to connect to your Web server. If clients are using dial-up or slow connections, and you set this value too low, they will not be able to connect.

To overcome this problem, WebSphere Application Server V5.0 offers an option within the plug-in configuration file that allows you to bypass the operating system timeout.

It is possible to change an attribute to the Server element called `ConnectTimeout`, which makes the plug-in use a non-blocking connect. Setting `ConnectTimeout` to a value of zero (default) is equal to not specifying the `ConnectTimeout` attribute, that is, the plug-in performs a blocking connect and waits until the operating system times out. Set this attribute to an integer value greater than zero to determine how long the plug-in should wait for a response when attempting to connect to a server. A setting of 10 means that the plug-in waits for 10 seconds to time out.

To determine what setting should be used, you need to take into consideration how fast your network and servers are. Complete some testing to see how fast your network is, and take into account peak network traffic and peak server usage. If the server cannot respond before the `ConnectTimeout`, the plug-in will mark it as down.

Since this setting is determined on the Server tag, you can set it for each individual cluster member. For instance, you have a system with four cluster members, two of which are on a remote node. The remote node is on another subnet and it sometimes takes longer for the network traffic to reach it. You might want to set up your cluster as shown in Example 5-21.

*Example 5-21 ConnectTimeout Server attribute in plugin-cfg.xml file*

---

```
<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="PluginCluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
  <Server CloneID="v544d031" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="8" MaxConnections="-1" Name="was1node_PluginMember1"
WaitForContinue="false">
    <Transport Hostname="app1.itso.ibm.com" Port="9088" Protocol="http"/>
  </Server>
```



```

<Server CloneID="v544d0o0" ConnectTimeout="20" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="was2Node_PluginMember2"
WaitForContinue="false">
  <Transport Hostname="app2.itso.ibm.com" Port="9086" Protocol="http"/>
</Server>
<Server CloneID="v54deevk" ConnectTimeout="10" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="-1" Name="was3node_PluginMember3"
WaitForContinue="false">
  <Transport Hostname="app3.itso.ibm.com" Port="9093" Protocol="http"/>
</Server>
<PrimaryServers>
  <Server Name="was1node_PluginMember1"/>
  <Server Name="was2Node_PluginMember2"/>
  <Server Name="was3node_PluginMember3"/>
</PrimaryServers>
</ServerCluster>

```

---

PluginMember1 in Example 5-21 on page 208 is on the same machine as the Web server. This means that there is no need to use a non-blocking connect. PluginMember2 is on a remote server in a slower part of the network. So PluginMember2 has a higher ConnectTimeout setting to compensate for this. Finally, PluginMember3 is on a faster part of the network, so it is safer to set the ConnectTimeout to a lower value.

If a non-blocking connect is used, you will see a slightly different trace output. Example 5-22 shows what you see in the plug-in trace if a non-blocking connect is successful.

---

*Example 5-22 Plug-in trace when ConnectTimeout is set*

---

```

...
TRACE: ws_common: websphereGetStream: Have a connect timeout of 10; Setting
socket to not block for the connect
TRACE: errno 55
TRACE: RET 1
TRACE: READ SET 0
TRACE: WRITE SET 32
TRACE: EXCEPT SET 0
TRACE: ws_common: websphereGetStream: Reseting socket to block
...

```

---

## RetryInterval setting

There is a setting in the plug-in configuration file that allows you to specify how long to wait before retrying a server that is marked as down. This is useful in avoiding unnecessary attempts when you know that server is unavailable. The default is 60 seconds.

This setting is specified in the `ServerCluster` element using the `RetryInterval` attribute. An example of this in the `plugin-cfg.xml` file is:

```
<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="PluginCluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="120">
```

This would mean that if a cluster member were marked as down, the plug-in would not retry it for 120 seconds.

There is no way to recommend one specific value; the value chosen depends on your environment. For example, if you have numerous cluster members, and one cluster member being unavailable does not affect the performance of your application, then you can safely set the value to a very high number.

Alternatively, if your optimum load has been calculated assuming all cluster members to be available or if you do not have very many, then you will want your cluster members to be retried more often to maintain the load.

Also, take into consideration the time it takes to restart your server. If a server takes a long time to boot up and load applications, then you will need a longer retry interval.

## MaxConnections setting

All requests to the application servers flow through the HTTP Server plug-in. The application server selection logic in the plug-ins has been enhanced so that it takes into account the number of pending connections to the application server. The `MaxConnections` attribute is used to specify the maximum number of pending connections to an application server that can be flowing through a Web server process at any point in time.

The plug-in config file has a new attribute `MaxConnections` for the `Server` definition as shown in Example 5-23. This feature is disabled if the value is set to -1 or 0. By default, the value is set to -1. The attribute can be set to the maximum number of pending connections to the application server through the plug-in.

*Example 5-23 MaxConnections Server attribute in plugin-cfg.xml file*

---

```
..
  <ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="PluginCluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
    <Server CloneID="v544d031" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="10" Name="waslnode_PluginMember1"
WaitForContinue="false">
        <Transport Hostname="app1.itso.ibm.com" Port="9088" Protocol="http"/>
    </Server>
```

```

        <Server CloneID="v544d0o0" ConnectTimeout="0" ExtendedHandshake="false"
LoadBalanceWeight="2" MaxConnections="10" Name="was2Node_PluginMember2"
WaitForContinue="false">
        <Transport Hostname="app2.itso.ibm.com" Port="9086" Protocol="http"/>
    </Server>
    <PrimaryServers>
        <Server Name="was1node_PluginMember1"/>
        <Server Name="was2Node_PluginMember2"/>
    </PrimaryServers>
</ServerCluster>
..

```

---

For example, let the two application servers be fronted by one node running IBM HTTP Server. If the MaxConnections attribute is set to 10, then each application server could potentially get up to 10 pending connections.

If the number of pending connections reaches the maximum limit of the application server, then it is not selected to handle the current request. If no other application server is available to serve the request, HTTP response code 503 (Service unavailable) is returned to the user.

This can be seen in the plugin trace listing in Example 5-24.

---

*Example 5-24 Plug-in trace when MaxConnections is set*

---

```

..
### When the request comes, it is pended.
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 0,
maxConnectionsCount 10.
TRACE: ws_server_group: lockedServerGroupUseServer: Server
was1node_PluginMember1 picked, weight 0.
TRACE: ws_server_group: serverGroupIncrementConnectionCount: Server
was1node_PluginMember1 picked, pendingConnectionCount 1 totalConnectionsCount
1.
TRACE: ws_common: websphereFindTransport: Finding the transport

### When the next request comes, it is pended again.
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 1,
maxConnectionsCount 10.
TRACE: ws_server_group: lockedServerGroupUseServer: Server
was1node_PluginMember1 picked, weight 0.
TRACE: ws_server_group: serverGroupIncrementConnectionCount: Server
was1node_PluginMember1 picked, pendingConnectionCount 2 totalConnectionsCount
2.
TRACE: ws_common: websphereFindTransport: Finding the transport

### When the number of pending connections reaches the MaxConnections(in this
case, 10), the HTTP response code 503 is returned to the user.

```

```

TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first
primary server
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 10,
maxConnectionsCount 10.
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of
was1node_PluginMember1, ignoreWeights 0, markedDown 0, retryNow 0, wlbAllows 1
reachedMaxConnectionsLimit 1
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 10,
maxConnectionsCount 10.
WARNING: ws_server_group: serverGroupCheckServerStatus: Server
was1node_PluginMember1 has reached maximum connections and is not selected
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next
primary server
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 10,
maxConnectionsCount 10.
STATS: ws_server_group: serverGroupCheckServerStatus: Checking status of
was2Node_PluginMember2, ignoreWeights 0, markedDown 0, retryNow 0, wlbAllows 1
reachedMaxConnectionsLimit 1
TRACE: ws_server: serverHasReachedMaxConnections: currentConnectionsCount 10,
maxConnectionsCount 10.
WARNING: ws_server_group: serverGroupCheckServerStatus: Server
was2Node_PluginMember2 has reached maximum connections and is not selected
ERROR: ws_server_group: serverGroupNextRoundRobinServer: Failed to find a
server; all could be down or have reached the maximum connections limit
WARNING: ws_common: websphereFindServer: Application servers have reached
maximum connection limit
ERROR: ws_common: websphereWriteRequestReadResponse: Failed to find a server
ERROR: ESI: getResponse: failed to get response: rc = 8
TRACE: ESI: esiHandleRequest: failed to get response
TRACE: ESI: esiRequestUrlStackDestroy
TRACE: ESI: esiRequestPopUrl: '/wlm/beenthere'
TRACE: ESI: esiUrlDestroy: '/wlm/beenthere'
ERROR: ws_common: websphereHandleRequest: Failed to handle request
TRACE: ws_common: websphereCloseConnection
TRACE: ws_common: websphereEndRequest: Ending the request
..

```

---

As mentioned earlier, when the plug-in detects that there are no application servers available to satisfy the request, HTTP response code 503 (Service unavailable) is returned. This response code appears in the Web server access log, as shown in Example 5-25.

*Example 5-25 HTTP Server access log example*

---

```

[08/Dec/2003:14:08:03 -0500] "GET /wlm/beenthere HTTP/1.0" 503 419
[08/Dec/2003:14:08:03 -0500] "GET /wlm/beenthere HTTP/1.0" 503 419
[08/Dec/2003:14:08:03 -0500] "GET /wlm/beenthere HTTP/1.0" 503 419

```

---

This feature helps you to better load balance the application servers fronted by the plug-in. If application servers are overloaded, the plug-in will automatically skip these application servers and try the next available application server.

This feature solves the main problem of application servers taking a long time to respond to requests. It is achieved by throttling the number of connections going to the application server through the plug-in.





## EJB workload management

The Enterprise JavaBeans (EJB) specification is a foundation of the Java 2 Platform Enterprise Edition (J2EE). Vendors use this specification to implement an infrastructure in which EJBs can be deployed, and use a set of services such as distributed transactions, security, life-cycle management, and database connectivity. IBM WebSphere Application Server Network Deployment V5.x provides another service for EJBs: *workload management* (WLM).

In this chapter, the following topics are discussed:

- ▶ Enabling EJB workload management
- ▶ EJB types and workload management
- ▶ Naming and name spaces
- ▶ How EJBs participate in workload management
- ▶ EJB server selection policy
- ▶ EJB workload management behavior
- ▶ EJB workload management failover
- ▶ Backup Cluster support

## 6.1 Enabling EJB workload management

Since EJB workload management means sharing the load among multiple application servers, this functionality is not available in IBM WebSphere Application Server Base V5.1, or IBM WebSphere Application Server for Developers V5.1, because these can run only one application server each.

In IBM WebSphere Application Server Network Deployment V5.x, workload management for EJBs is enabled automatically when a cluster is created within a cell. There is no need for a special configuration to enable it. Workload management uses a WLM plug-in in the ORB (Object Request Broker) to dispatch the load among the application servers (cluster members) of the cluster.

Figure 6-1 shows how workload management is handled by IBM WebSphere Application Server Network Deployment V5.x.

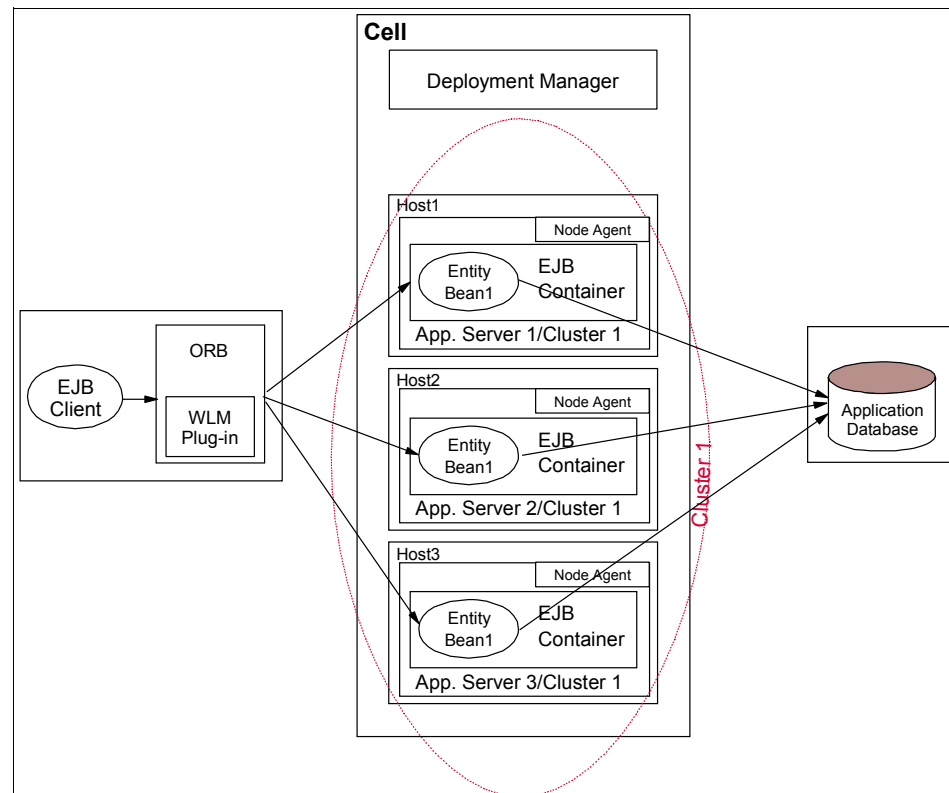


Figure 6-1 IBM WebSphere Application Server V5.1 EJB workload management



IBM WebSphere Application Server Network Deployment V5.x uses the concept of *cell*, *cluster*, and *cluster members*, as described in 1.3.3, “Workload management using WebSphere clustering” on page 16 to identify which application servers participate in workload management. Requests from clients are distributed among the cluster members’ EJB containers within the cluster.

**Note:** The WebSphere-supplied Java Runtime Environment (JRE) is required for both the runtime and any remote Java clients.

## 6.2 EJB types and workload management

The workload management service provides load balancing for the following types of enterprise beans:

- ▶ Homes of entity or session beans
- ▶ Instances of entity beans
- ▶ Instances of stateless session beans

As shown in Figure 6-2, the only type of EJB references not subject to load distribution through EJB WLM are instances of a given stateful session bean. See 6.2.2, “Stateful session beans” on page 218 for details.

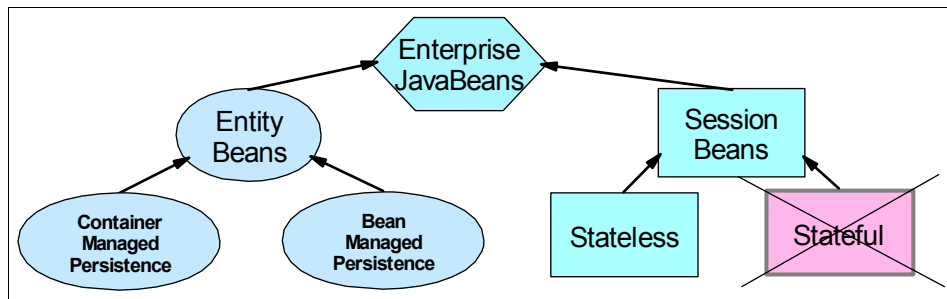


Figure 6-2 Enterprise beans that participate in workload management

### 6.2.1 Stateless session beans

By definition, a stateless session bean maintains no state information. Each client request directed to a stateless session bean is independent of the previous requests that were directed to the bean. The EJB container maintains a pool of instances of stateless session beans, and provides an arbitrary instance of the appropriate stateless session bean when a client request is received. Requests can be handled by any stateless session bean instance in any cluster member of a cluster, regardless of whether the bean instance handled the previous client requests.

## 6.2.2 Stateful session beans

A stateful session bean is used to capture state information that must be shared across multiple consecutive client requests that are part of a logical sequence of operations. The client must obtain an EJB object reference to a stateful session bean to ensure that it is always accessing the same instance of the bean.

WebSphere Application Server currently supports the clustering of stateful session bean home objects among multiple application servers. However, it does not support the clustering of a specific instance of a stateful session bean. Each instance of a particular stateful session bean can exist in just one application server and can be accessed only by directing requests to that particular application server. State information for a stateful session bean cannot be maintained across multiple application server cluster members.

**Note:** Even though stateful session beans are not workload managed themselves, WLM can still be achieved when the homes are evenly distributed. It is only after the bean is created that everything will be performed on the same cluster member.

## 6.2.3 Entity beans

An entity bean represents persistent data. Most external clients access entity beans by using session beans, but it is possible for an external client to access an entity bean directly. The information contained in an entity bean is not usually associated with a session or with the handling of one client request or series of client requests. However, it is common for a client to make a succession of requests targeted at the same entity bean instance. It is also possible for more than one client to independently access the same entity bean instance within a short time interval. The state of an entity bean must therefore be kept consistent across multiple client requests.

For entity beans, the concept of a session is replaced by the concept of a transaction. An entity bean is instantiated in a container for the duration of the client transaction in which it participates. All subsequent accesses to that entity bean within the context of that transaction are performed against that instance of the bean in that particular container. The container needs to maintain state information only within the context of that transaction. The workload management service uses the concept of *transaction affinity* (as defined in 6.6.4, “Transaction affinity” on page 246) to direct client requests. After an EJB server entity bean is selected, client requests are directed towards it for the duration of the transaction.

Between transactions, the state of the entity bean can be cached. The WebSphere V5.x EJB container supports option A, option B, and option C caching:

► Option A caching

WebSphere Application Server assumes that the entity bean is used within a single container. Clients of that bean must direct their requests to the bean instance within that container. The entity bean has exclusive access to the underlying database, which means that the bean cannot be clustered or participate in workload management if option A caching is used.

► Option B caching

The bean instance remains active (so it is not guaranteed to be made passive at the end of each transaction), but it is always reloaded from the database at the start of each transaction. A client can attempt to access the bean and start a new transaction on any container that has been configured to host that bean.

► Option C caching (default)

The entity bean is always reloaded from the database at the start of each transaction and passivated at the end of each transaction. A client can attempt to access the bean and start a new transaction on any container that has been configured to host that bean.

Entity beans can participate in workload management as long as the server reloads the data into the bean at the start of each transaction, assuming that transactional affinity is in place. Guaranteed passivation at the end of each transaction is not a requirement for a bean to participate in workload management. Hence, option B and option C caching are both compatible with workload management, but option A caching is not.

Table 6-1 provides a summary of the EJB caching options.

*Table 6-1 EJB caching options*

Option	Activate at must be set to	Load at must be set to
A	Once	Activation
B	Once	At start of transaction
C (default)	At start of transaction	At start of transaction

## 6.3 Naming and name spaces

Naming is used by clients of IBM WebSphere Application Server V5.x applications to obtain references to objects related to those applications, such as Enterprise JavaBeans (EJB) homes.

These objects are bound into a mostly hierarchical structure, referred to as a *name space*. An InitialContext is used to access objects in the name space. To obtain an InitialContext a bootstrap server and port can be supplied. If one is not supplied, then default values will be used specific to the client type and its environment.

WebSphere Application Server name servers are an implementation of the CORBA CosNaming interface.

**Important:** In WebSphere Application Server V4.0 there was only one name space for an entire Administrative Domain (now known as a cell). However this has changed with IBM WebSphere Application Server Network Deployment V5. Now instead of one name space for the entire cell, each application server, Node Agent, and the Deployment Manager has its own name space.

However, the separate name spaces are *federated* (through context links between the name spaces) to form a single logical name space for the cell.

The default port value for the bootstrap server provider URL now defaults to port 2809 as required by the CORBA 2.3 specification. This is different from WebSphere Application Server V4.0, Advanced Edition which defaulted to port 900.

InitialContext requests participate in workload management when the provider URL is a clustered resource (cluster member) and they do not when they are not a clustered resource.

More information about naming and name spaces can be found in the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195. Chapter 4 explains the concept in detail.

### 6.3.1 Looking up an EJB home with JNDI examples

The name and method used to look up an object depends on whether or not the application is running in a container. This section describes some methods and best practices of performing a JNDI lookup for particular client environments.

## Perform lookup in an EJB or Web container in the same cell

When an application that is running within an EJB or Web container wants to perform a lookup of an EJB in the same cell, then best practice is to use an EJB reference in the application code and an InitialContext object with no parameters.

An EJB reference is a method of delegating the JNDI name resolution of an EJB from the application to the deployment descriptor. Using the prefix `java:comp/env/` on the front of the JNDI name informs the container that this particular JNDI lookup resolves to a reference specified in the deployment descriptor. This removes the reliance on either hard coded JNDI names in the application code or reliance on external properties files.

Only when running in either a Web, EJB, or J2EE application client container can references be used as it is the job of the container to resolve those references using the deployment descriptors.

The binding of an EJB reference to a real JNDI name is specified in the deployment descriptor and so can be altered during or after deployment using the WebSphere Administrative Console. See “Perform lookup in an EJB or Web container in the same cell” on page 221.

The InitialContext object used to perform the lookup does not need any parameters if the target EJB is in the same cell. Leaving the InitialContext object empty will mean the local application server’s naming service in which the client is running will be used for lookups. This is because all the name spaces throughout the cell are linked, so a fully qualified JNDI name will locate the EJB home.

So, if performing the lookup from within a Web or EJB container to an EJB that is in another process in the same cell, then the JNDI name needs to be either fully qualified with the node and server that contains the EJB or, if the EJB is on a clustered server, the cluster name. This means the JNDI name `ejb/myEJB` has to become one of the following to work:

- ▶ `cell/nodes/Node1/servers/Server1/ejb/MyEJB`
- ▶ `cell/clusters/MyCluster/ejb/MyEJB`

Example 6-1 shows a lookup of an EJB home when the client is running in an EJB or Web container that is looking up an EJB in the same cell. The EJB reference `java:comp/env/ejb/BeenThere` has been set to resolve to `cell/clusters/MyCluster/ejb/BeenThere` in the EJB deployment descriptor.

### *Example 6-1 Lookup of an EJB home*

---

```
// Get the initial context
import java.util.Hashtable;
import javax.naming.Context;
```

```

import javax.naming.InitialContext;

Context initialContext = new InitialContext();

// Look up the home interface using the JNDI name
try {
    java.lang.Object ejbHome =
        initialContext.lookup("java:comp/env/ejb/BeenThere");
    beenThereHome = (BeenThereHome)javax.rmi.PortableRemoteObject.narrow(
        (org.omg.CORBA.Object) ejbHome, BeenThereHome.class);
}
catch (NamingException e) { // Error getting the home interface
    ...
}

```

---

Delegating the details of both the JNDI name and the InitialContext location to the container makes the application code much more portable between different environments.

In “Server cluster with fault-tolerant initial context” on page 227 there is a discussion about fault tolerant name lookups, this does not apply here as if the local naming service is unavailable then the application server itself will probably not be running.

### ***Mapping EJB references to enterprise beans***

With IBM WebSphere Application Server Network Deployment V5.x, you can map the EJB references to enterprise beans after deployment of your application. System administrators can bind EJB references specified by the application developer to a required EJB home in a target operational environment by setting the JNDI name value using the Administrative Console as follows:

1. Select your application from **Applications--> Enterprise Applications--> <application\_name>**.
2. Select **Map EJB references to beans** from the Additional Properties section.
3. Enter the JNDI name of your Enterprise Bean(s) in the JNDI Name column as shown in Figure 6-3 on page 223.
4. Stop and restart all application server(s) where your application is installed.

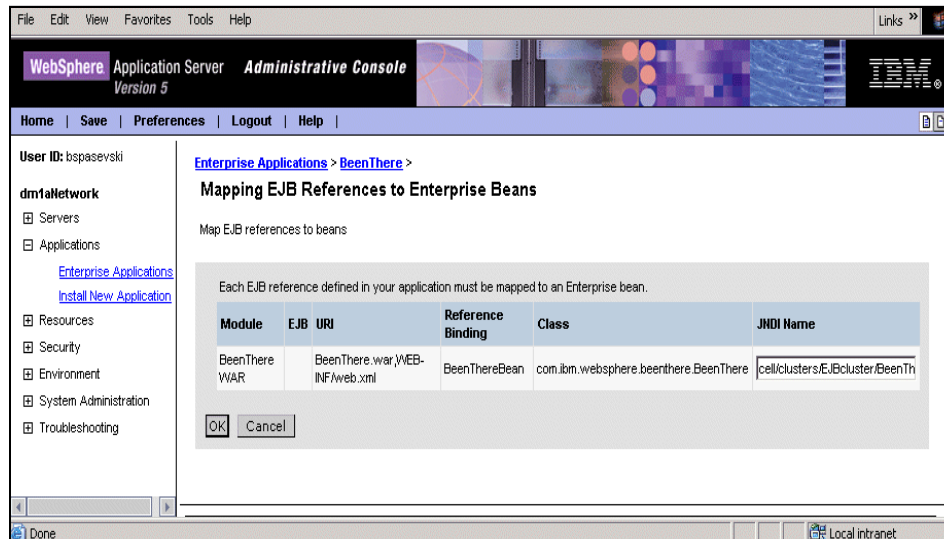


Figure 6-3 Mapping EJB references to Enterprise Beans

**Note:** Mapping of EJB references to Enterprise Beans can also be performed at deployment time of your application. This is done in the step titled “Map EJB references to beans”, as shown in Figure 6-4 on page 224.

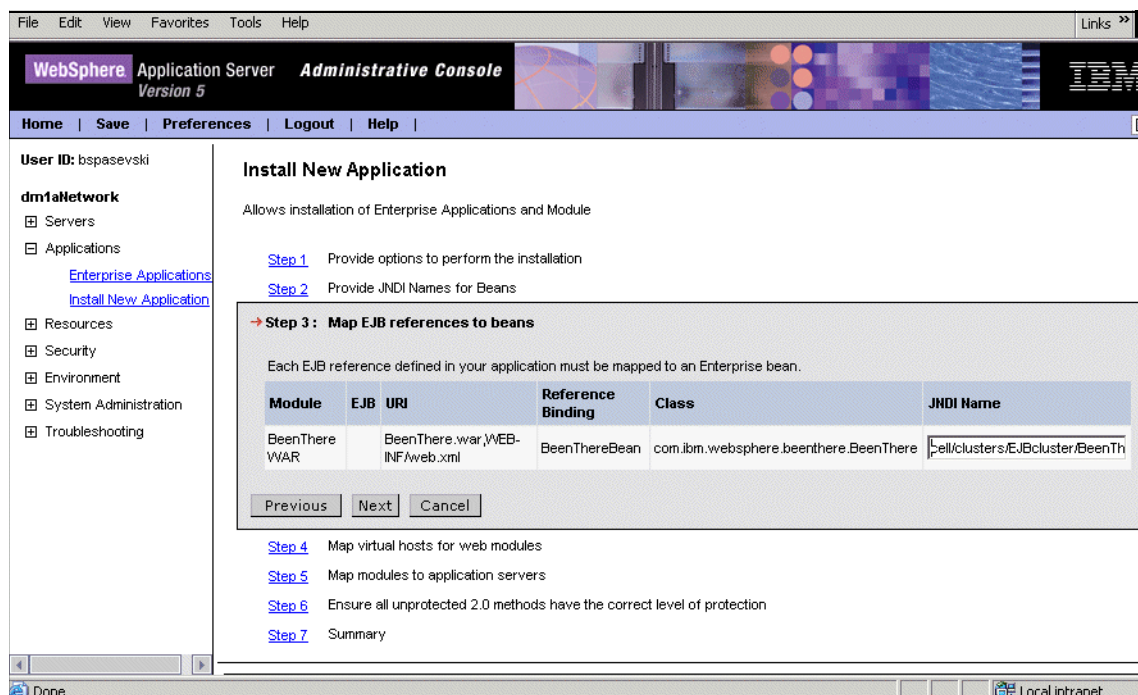


Figure 6-4 Deployment time, mapping EJB references to Enterprise Beans

## Perform lookup in an EJB / Web container outside the cell

It is slightly different when performing a JNDI lookup from an EJB or Web container for an EJB that is not within the same cell. EJB references can still be used but the local name server is not able to locate the EJB home. In this situation a provider URL is needed when creating the InitialContext object that points to the name server that does know about the EJB. This makes this scenario fall somewhere between a J2EE application client container and a stand alone client. EJB references can be used but the container cannot be relied to locate a name server that can find the EJB.

## Perform lookup in a J2EE application client container

The J2EE application client container provides some similar facilities to an EJB or Web container in terms of naming. The client container will do EJB reference lookups so the code in Figure 6-1 on page 221 would also work here. It uses the application clients deployment descriptor to do the binding of a reference to a real JNDI name.

As the calling code is actually running outside of an application server process it still needs to be able to resolve the InitialContext object to the correct location to



lookup the EJB. Using the J2EE application client container removes the need to hard code the location of the application server that is providing the name service. The code in Figure 6-1 on page 221 does a call without passing any parameters to the InitialContext. When running in the J2EE application client container this is resolved to the JNDI name server that is specified when launching the client container. For example, using the command:

```
launchclient ClientApp.ear -CCBootstrapHost=app1  
-CCBootstrapPort=2809
```

will launch the J2EE client contained within the EAR file. When an InitialContext object is created with no parameters the J2EE application client container will resolve this to the name service running on app1 on port 2809.

This means that although the code is written the same as for running in an EJB or Web container, the actual resolution of the correct name service relies on parameters passed to the J2EE application client container at initialization.

When running in this container it is important that the client is able to resolve the process where the name server is running. If it is unable to do this then the client will fail to initialize.

To get around this issue it is possible to use a corbaloc provider URL to specify multiple locations for name servers that can be used to perform the JNDI lookup. This alternative command will run the client:

```
launchclient ClientApp.ear  
-CCproviderURL=corbaloc::app1:9813,:app2:9814
```

When it comes to performing the InitialContext lookup, the container has a choice of name servers, either the application server on app1 listening on port 9813 or the application server on app2 listening on port 9814.

This list of locations is not processed in any particular order so there is no way to guarantee which server will be used, but remember this is just for looking up the EJB home object. Once that is obtained, normal EJB workload management decides which server in a cluster should actually be used. Should one of these name servers fail to respond then the other will be used, removing the single point of failure.

If the target EJB is running in a cluster then this is the recommended method of looking up EJB homes when running in a J2EE application client container, as it provides failover.

When populating this list remember that whichever name server you point the client at has to be able to resolve the JNDI name the EJB reference resolves to.

The possible options for fully qualifying the JNDI name are discussed at the end of “Perform lookup in an EJB or Web container in the same cell” on page 221.

## Perform lookup in a Java stand alone client

Applications that do not run in a container cannot use `java:lookup` names because it is the container that configures the `java:lookup` name space for the application. Instead, an application of this type must look up the object directly from the name server. Also, the stand alone client cannot rely on a container to resolve the location of the name server as it is running outside of one. When this is the case the name server location needs to be specified in the application code.

As the location of the name server and the JNDI name of the EJB are environment specific, the stand alone client should obtain the necessary values to pass to the `InitialContext` from an external resource, like a properties file.

In the following sections, we show three examples of JNDI lookups to application servers running the target EJB. These examples do not use a properties file. There is one example to connect to a:

- ▶ Single server
- ▶ Server cluster
- ▶ Server cluster with fault-tolerant initial context

### ***Single server***

Example 6-2 shows the lookup of an EJB home that is running in the single server, `Server1`, configured in the node `app1`. In this example there is only one server so there is no possibility of specifying multiple name servers. In this example the name server is running on the same application server as the target EJB, so the JNDI name does not need to be fully qualified.

#### *Example 6-2 Single server - lookup EJB home*

---

```
// Get the initial context
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc::app1:2809");
Context initialContext = new InitialContext(env);
// Look up the home interface using the JNDI name
try {
    java.lang.Object ejbHome = initialContext.lookup(
"ejb/BeenThere");
    BeenThereHome = (BeenThereHome)javax.rmi.PortableRemoteObject.narrow(
        (org.omg.CORBA.Object) ejbHome, BeenThereHome.class);
}
catch (NamingException e) { // Error getting the home interface
```

```
    ...  
}
```

---

### ***Server cluster***

Example 6-3 shows the lookup of an EJB home that is running in the cluster, EJBcluster. The name can be resolved if any of the cluster members are running. As this is a stand alone client, the location of the name service still needs to be specified.

#### *Example 6-3 Server cluster - lookup EJB home*

---

```
// Get the initial context  
Hashtable env = new Hashtable();  
env.put(Context.INITIAL_CONTEXT_FACTORY,  
"com.ibm.websphere.naming.WsnInitialContextFactory");  
env.put(Context.PROVIDER_URL, "corbaloc::appl:2809");  
Context initialContext = new InitialContext(env);  
// Look up the home interface using the JNDI name  
try {  
    java.lang.Object ejbHome = initialContext.lookup(  
        "cell/clusters/EJBcluster/ejb/BeenThere");  
    beenThereHome = (BeenThereHome)javax.rmi.PortableRemoteObject.narrow(  
        (org.omg.CORBA.Object) ejbHome, BeenThereHome.class);  
}  
catch (NamingException e) { // Error getting the home interface  
    ...  
}
```

---

### ***Server cluster with fault-tolerant initial context***

In the previous example, the EJB is running on clustered servers but still relies on one server for name service lookups. Example 6-4 shows how to obtain the initial context via a fault-tolerant provider URL.

#### *Example 6-4 Cluster - lookup EJB home via fault-tolerant provider URL & specify path*

---

```
// Get the initial context  
Hashtable env = new Hashtable();  
env.put(Context.INITIAL_CONTEXT_FACTORY,  
"com.ibm.websphere.naming.WsnInitialContextFactory");  
env.put(Context.PROVIDER_URL, "corbaloc::appl:2809,app2:2809");  
Context initialContext = new InitialContext(env);  
  
// Look up the home interface using the JNDI name  
try {
```

```
java.lang.Object ejbHome =  
initialContext.lookup("cell/clusters/EJBcluster/BeenThere");
```

---

This is fault tolerant, so if one process running a name server goes down, another can be used, and the cluster can be accessed via the appropriate cell-based path in the lookup.

## 6.4 How EJBs participate in workload management

In this section, we examine how EJBs participate in workload management through the following stages:

- ▶ Initial request
- ▶ Subsequent requests
- ▶ Cluster run state changes

### 6.4.1 Initial request

When accessing an EJB there are two groupings of clients, cluster aware and cluster unaware. Cluster aware clients are those that are running within an IBM ORB and therefore have access to the WLM information about WebSphere Application Server clusters. For more information about cluster aware and cluster unaware clients see “EJB WLM routing” on page 368.

These steps describe what happens when using a cluster aware client to access an EJB:

1. First, the client has to retrieve the initial context. This varies between client types as discussed in “Looking up an EJB home with JNDI examples” on page 220.
2. Next, the client needs to look up the EJB home based on the JNDI name, for example:

```
Object home = initContext.lookup("java:comp/env/BeenThere");  
BeenThereHome beentherehome =  
    (BeenThereHome) narrow(home, BeenThereHome.class);
```

**Note:** This example uses an EJB reference, `java:comp/env/BeenThere`. As discussed in “Perform lookup in an EJB or Web container in the same cell” on page 221, this EJB reference must be bound to the fully qualified JNDI name of the deployed EJB, for example:

```
cell/clusters/EJBcluster/BeenThere.
```

Using EJB references would not be possible in a stand alone EJB client as it is not running in a container.

3. The client needs to create or retrieve an EJB object, using the create method or finders of the home interface, for example:

```
BeenThere beenThere = beentherehome.create();
```

4. Once the EJB object has been created, you can invoke methods from the remote interface, for example:

```
Hashtable envInfo = beenThere.getRuntimeEnvInfo();
```

We will call these four steps the initial request from the EJB client. Let us see in detail what is happening from a workload management point of view, using Figure 6-5 on page 230:

1. The new InitialContext request goes through the ORB (Object Request Broker). This returns a JNDI context object.
2. The lookup on the context returns a home object of the BeenThere bean. This is an indirect IOR (Interoperable Object Reference), that is, it points to the Location Service Daemon (LSD) on the local Node Agent.
3. The first request goes to the LSD and the LSD selects one of the cluster members by using the WLM plug-in in the LSD.
4. The LSD returns a direct IOR to the specific cluster member.
5. The request is forwarded to the cluster member that the LSD selected.
6. Upon successful completion of the request, the response contains the cluster configuration information. WLM plug-in stores the cluster configuration information and uses it for subsequent requests.

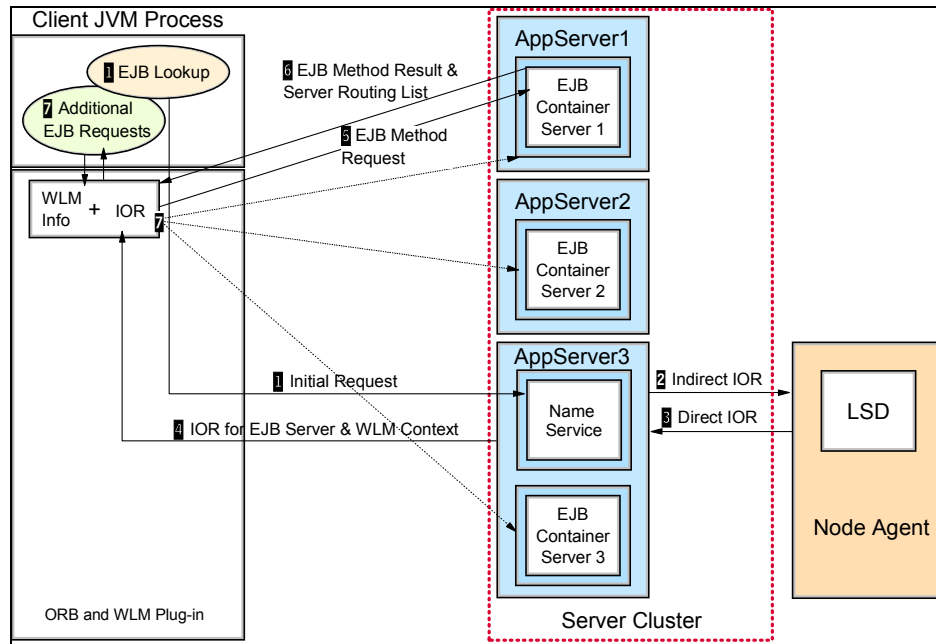


Figure 6-5 EJB workload management

## 6.4.2 Subsequent requests

Now let us look at what happens with subsequent EJB requests:

1. Subsequent requests from the client to a remote method go through the ORB as well.
2. The ORB asks the WLM plug-in for the IOR of the server in order to process the request.
3. Based on the workload management policy, process affinity, and transaction affinity (see 6.5, “EJB server selection policy” on page 232), the WLM plug-in returns the IOR of the next target.
4. The ORB invokes the remote method on the selected server.

## 6.4.3 Cluster run state changes

To conclude this section, we examine how the WLM plug-in is informed of changes to the cluster configuration:

1. Changes are made to the cluster configuration, such as adding and starting a fourth cluster member to the example.

2. The Deployment Manager pushes the changes to the Node Agents which in turn push those changes to all cluster members.
3. Meanwhile, the EJB client is still performing requests on the cluster.
4. With each request for a remote component, information about the cluster is returned in the response from the cluster member.

If the cluster has been modified since the last request from this client. The WLM plug-in will update its cluster data using the new data returned in the response.

5. The EJB client makes another method call to the remote interface.
6. The ORB that handles the request asks the WLM plug-in for the IOR of the server to contact.
7. The WLM plug-in returns the IOR of the next target, based on the workload management policy, process affinity, and transaction affinity (see 6.5, “EJB server selection policy” on page 232), and the request can be processed by the ORB.

**Important:** A change in the selection policy does not cause the cluster information to be sent to a client in response to a request. The WLM plug-in will continue using the selection policy defined at the first request. If the cluster topology or the number of cluster members started is changed, the WLM plug-in will get the new selection policy as part of the new cluster configuration in the response.

Each Node Agent monitors the availability of the application servers running on it. Should one of the cluster member servers be stopped then this constitutes a change in run state for the cluster and the Node Agent will notify the Deployment Manager of that change. This information is then pushed out again as was described in step 2.

The Node Agent knows if an application server is still running by pinging it intermittently. If an application server should fail then the Node Agent will no longer receive responses on its ping messages. In this scenario, the Node Agent will notify the Deployment Manager of the run state change and again this information will be pushed out to the cluster.

If a complete failure of a node occurs, then the Deployment Manager itself will not receive responses from its ping of the Node Agent and the clusters' configuration will be updated.

From this it is possible to see that for the initial request and for workload management to work efficiently, the Deployment Manager and all Node Agents must be up and running. Failure in the Deployment Manager could mean that

these run state configuration changes will not get notified to an EJB client in a timely manner.

## 6.5 EJB server selection policy

An EJB server selection policy defines how clients (such as servlets, stand-alone Java clients, or other EJBs) choose among EJB cluster members (instances). EJB workload management offers the following selection policies:

- **Server weighted round robin routing**

The server weighted round robin routing will select the next currently available cluster member. The policy ensures a distribution based on the set of server weights that have been assigned to the members of a cluster. For example, if all servers in the cluster have the same weight, the expected distribution for the cluster would be that all servers receive the same number of requests. If the weights for the servers are not equal, the distribution mechanism will send more requests to the higher weight value servers than the lower weight value servers. The policy ensures the desired distribution, based on the weights assigned to the cluster members.

The server weight value defaults to 2 for each member of the cluster and is maintained in the cluster routing table.

**Tip:** When setting the server weight values for your cluster members, you should utilize low values to avoid load variations.

For example, you would be better off by setting server weights to 2 and 5 versus 8 and 20 so the refresh will occur more often and thus the server with the weight of 8 won't have to sit idle while 12 requests go to the server with a weight of 20. It would only sit idle for three requests instead of 12 requests.

Valid values for the weights range from 0 to 20.

If a particular EJB server instance is stopped or otherwise unavailable, that instance is skipped (no attempt is made to select it) until it can be verified as being back in service.

The ORB in the EJB client has a routing table for each cluster. The routing table is re-calculated for every new request that comes in. There are no



additional requests to an application server once its outstanding request ratio has reached its server weight value, but there are exceptions to this rule:

- Transaction affinity

Within a transaction, the first time an EJB server (entity bean) is chosen, the prevailing selection policy for the cluster member is applied. After the EJB server is selected, it remains bound for the duration of the transaction.

- Process affinity

When the Web container and the EJB container are configured in the same application server, the Web container never routes EJB requests to a container in a separate application server.

**Note:** Process affinity applies to servlet and EJB clients only. It does not affect stand-alone Java clients, because their EJB requests come from outside of the application server process.

- Prefer local (selected by default)

Along with the server weighted round robin routing, there is also a Prefer local policy.

Once the Prefer local policy is turned on, it is used for every cluster member in your cluster. Similarly, when you turn it off, it is off for every cluster member in your cluster.

With the Prefer local policy, the selection made by the WLM plug-in not only depends on the running cluster members, but also on the node (= machine) where the request comes from. The WLM plug-in only selects cluster members on the same node as the client, unless all local cluster members are unavailable.

The advantage of the Prefer local policy is that there are no network communications between the client and the EJB, so depending on the chosen topology, this policy can provide improved performance.

The client is the Java virtual machine (JVM) in which the code calling the EJB is running. This client might be a WebSphere process such as an application server running a Web container, or an application server running an EJB container.

When a servlet calls an EJB, the client is the application server that contains the servlet, and if the Prefer local policy is selected, the requests will go to the EJB container running on the same system. If EJB1 is calling EJB2, the client is the application that is running the EJB container containing EJB1. With the Prefer local policy, the requests will go to the same EJB container if EJB2 can

be found (see 6.6.3, “Process affinity” on page 246), or to another EJB container running on the same system.

This client might also be a Java virtual machine not running in WebSphere, such as a J2EE client application, or a stand-alone Java program accessing EJBs. For a J2EE client application, the Prefer local policy has no influence on the request distribution because the client runs on a remote workstation.

In the case of a Java program running on the same machine as WebSphere and using the WebSphere JRE and its ORB, the Prefer local policies will dispatch requests among EJB containers running on the same machine. EJB WLM requires the WebSphere ORB and its WLM plug-in. If non-WebSphere ORBs are used, then the Java client will not be able to participate in EJB WLM.

If there is no cluster member available on the local system (because of a failure, or because of the topology), the request will be dispatched to available cluster members following the server weighted round robin routing policy, as described in 6.6.1, “WLM behaviors using server weighted round robin” on page 240.

**Note:** Server weighted round robin routing replaces the random and round robin routing for EJB WLM that was used in WebSphere Application Server V4.0, Advanced Edition.

The Prefer local option is based on topology and pre-production test results. Naturally, the local host call will be quicker, and if you can put your clients (usually Web containers) on the same machines as your servers, the Prefer local option is a good choice. If you have clients only on a subset of your machines, then you will need to analyze the load distribution, since you will have client requests coming from remote machines as well as from the local machine.

Next we show how to configure server weighted round robin routing and Prefer local.

### 6.5.1 Server weighted round robin routing configuration

Server weights are an attribute of the cluster, not the cluster member. They are associated with the members of a cluster but they are only meaningful when set on the cluster.

To set the EJB workload management server weights for your cluster members:

1. Locate the cluster member by selecting **Servers -> Clusters -> <cluster\_name> -> Cluster members** in the WebSphere Administrative Console.
2. Select <member\_name> from the Cluster members page.
3. Select the **Configuration** tab from the Cluster members page.
4. Enter an appropriate value into the Weight field, as shown in Figure 6-6.
5. Click **Apply** and save the configuration changes.
6. Stop and start the Deployment Manager to change the weights in the configuration.

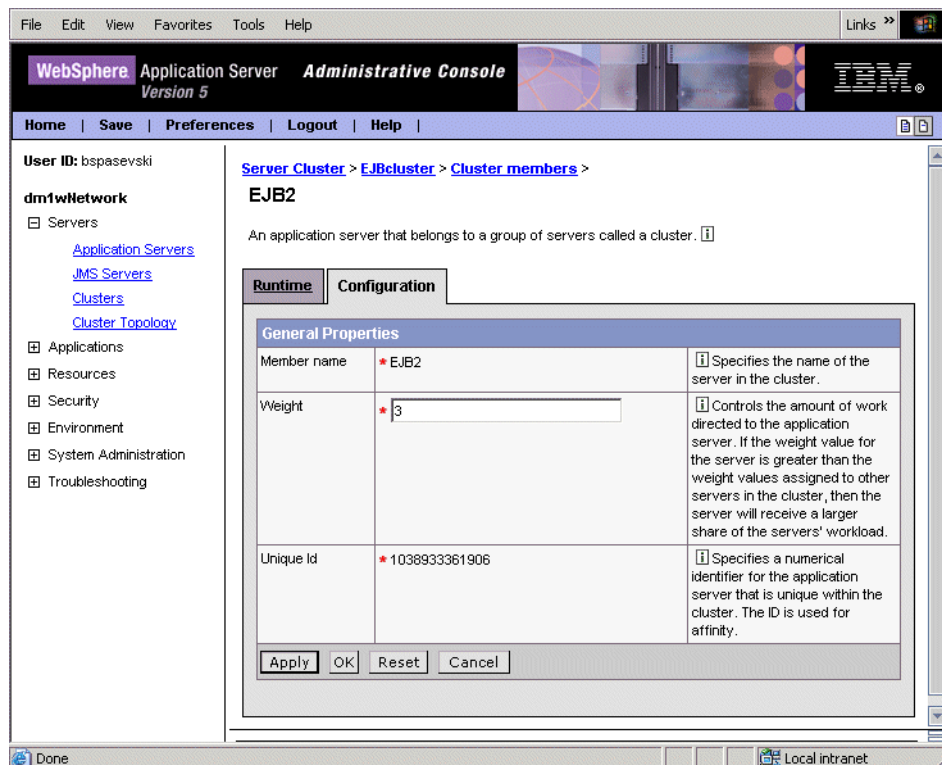


Figure 6-6 Workload management configuration weight value

## Runtime changes

To make a change to a cluster's server weights so that it only affects the current running system, that is, the weights will not be saved to the cluster's configuration, do the following:

1. Locate the cluster member by clicking **Servers -> Clusters -> <cluster\_name> -> Cluster members** in the WebSphere Administrative Console.
2. Select **<member\_name>** from the Cluster members page.
3. Select the **Runtime** tab from the Cluster members page.
4. Enter the right value into the Weight field, as shown in Figure 6-7.
5. Click **Apply** and save the changes.
6. Stop and start the application server to activate the changes.

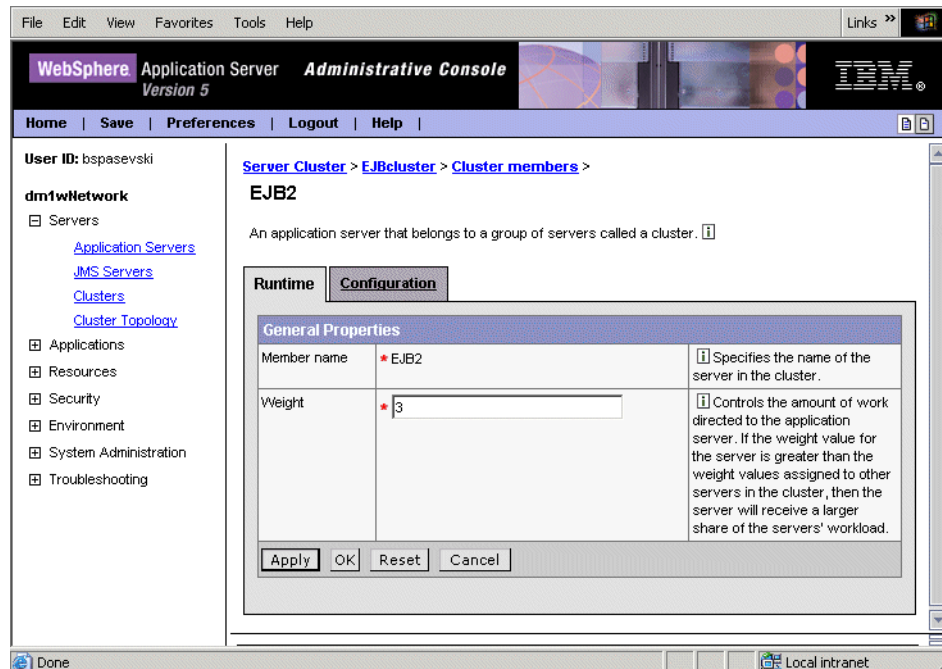


Figure 6-7 Workload management runtime weight value

**Note:** With IBM WebSphere Application Server Enterprise V5.1, the way the Runtime Weight value is applied varies from IBM WebSphere Application Server Network Deployment V5.1. An application server restart will not be required, which means the change to the weight value will be dynamically handled by the Deployment Manager.

## 6.5.2 Prefer local configuration

To activate the EJB workload management Prefer local option for a cluster:

1. Locate the cluster by clicking **Servers -> Clusters** in the WebSphere Administrative Console.
2. Select **<cluster\_name>** from the Server Cluster page.
3. Select the **Configuration** tab.
4. Check the **Prefer local** box as shown in Figure 6-8.
5. Click **Apply** and save the configuration changes.
6. Stop and start the Deployment Manager to activate Prefer local in your configuration.

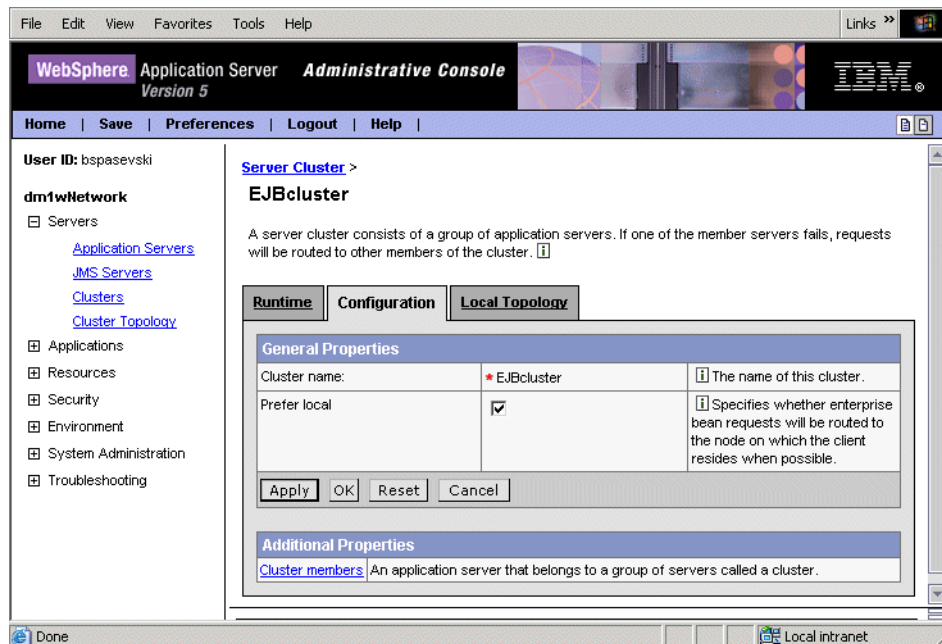


Figure 6-8 Workload management configuration Prefer local option

## Runtime changes

To activate the EJB workload management Prefer local option so that it only affects the current running system, that is, the Prefer local activation will not be saved to the cluster's configuration, do the following:

1. Locate the cluster by selecting **Servers -> Clusters** in the WebSphere Administrative Console.
2. Select **<cluster\_name>** from the Server Cluster page.
3. Select the **Runtime** tab.
4. Check the **Prefer local** box as shown in Figure 6-9.
5. Click **Apply** and save the changes.
6. Stop and start the application server(s) to activate your changes.

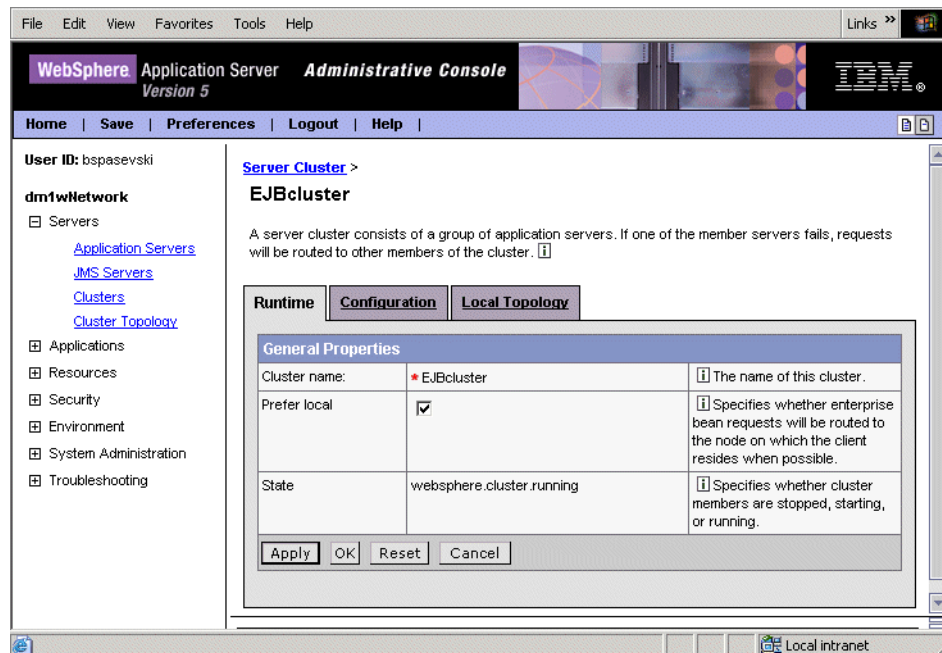


Figure 6-9 Workload management runtime Prefer local option

## 6.6 EJB workload management behavior

The BeenThere workload management demonstration can be used to view workload management behavior. BeenThere is composed of a servlet and a stateless session EJB. The purpose of BeenThere is to collect information about the execution environment. The servlet retrieves information from the Web container, and the EJB retrieves information from the EJB container. The information retrieved is the server name, Web or EJB container name, and Java process ID.

The BeenThere example can be used to demonstrate workload management behavior for a Web client. We added several EJBs and a J2EE application client to the example so we could more comprehensively demonstrate EJB WLM behavior.

**Note:** We used the topology shown in Figure 6-10 on page 239 to demonstrate the WLM policies. This topology is not intended to be used as is for a production environment. This topology was used so we could show all the various features of EJB workload management.

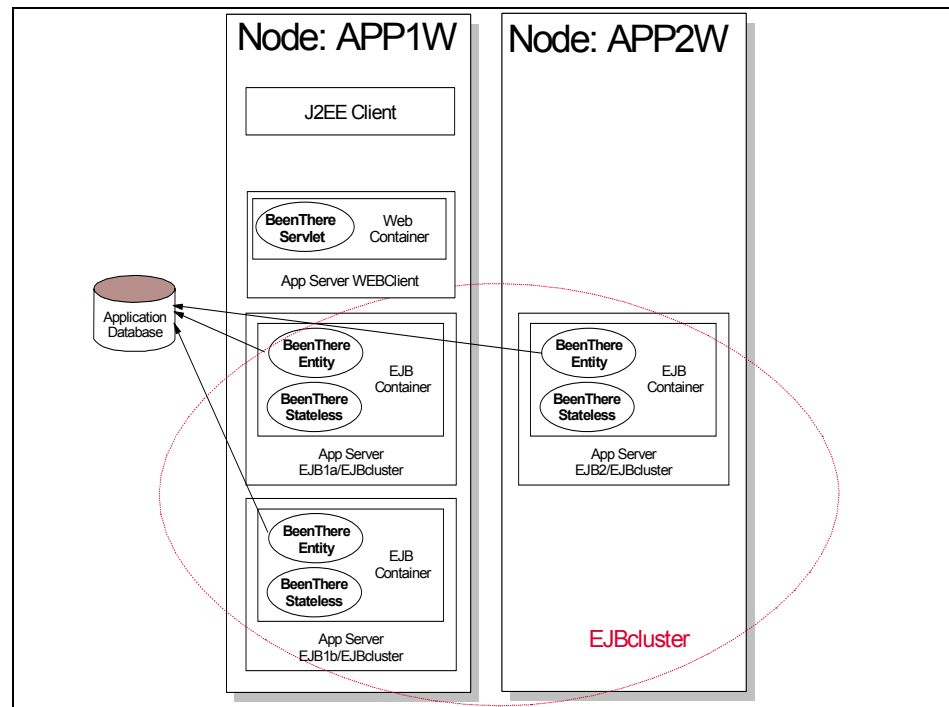


Figure 6-10 Sample topology for EJB workload management policies

Table 6-2 lists the cluster members weight values we used for our examples:

*Table 6-2 Weight values used for our examples*

Cluster member	Weight
EJB1a	2
EJB1b	2
EJB2	3

### 6.6.1 WLM behaviors using server weighted round robin

Assuming a request has no affinity, the server weighted round robin routing will select the next currently available cluster member, based on the distribution described by the weights for the cluster members.

**Important:** Whatever the server selection policy, process affinity and transaction affinity will always override the selection policy.

For example, in Table 6-2, the cluster weights describe a 2/7, 2/7, 3/7 request distribution. This means that if there are seven requests sent from a client, EJB1a will get two of the requests, EJB1b will also get two and EJB2 will get three. As you can see in Figure 6-11 on page 241, the EJB requests are distributed across all available cluster members in a repetitive order until cluster members' EJB1a and EJB1b server weight value reaches zero, at which point EJB2 receives an extra request. Remember, both EJB1a and EJB1b have the same server weight value of 2, where EJB2 has a server weight value of 3.



**BeenThere Workload Management Demonstration**

**Application Execution Results**

Iteration	Servlet Node	Servlet Server Name	Servlet PID	Bean Node	Bean Server Name	Bean PID
1	app1w	WEBClient	2224	app2w	EJB2	2064
2	app1w	WEBClient	2224	app1w	EJB1a	1628
3	app1w	WEBClient	2224	app1w	EJB1b	1544
4	app1w	WEBClient	2224	app2w	EJB2	2064
5	app1w	WEBClient	2224	app1w	EJB1a	1628
6	app1w	WEBClient	2224	app1w	EJB1b	1544
7	app1w	WEBClient	2224	app2w	EJB2	2064
8	app1w	WEBClient	2224	app2w	EJB2	2064
9	app1w	WEBClient	2224	app1w	EJB1a	1628
10	app1w	WEBClient	2224	app1w	EJB1b	1544
11	app1w	WEBClient	2224	app2w	EJB2	2064
12	app1w	WEBClient	2224	app1w	EJB1a	1628
13	app1w	WEBClient	2224	app1w	EJB1b	1544

Figure 6-11 Web client calling EJBs - round robin server selection

The output listing in Example 6-5 shows a similar behavior for our J2EE client application. EJB requests are distributed across all available cluster members in a repetitive order. Once cluster members' EJB1a and EJB1b server weight value reaches zero, EJB2 receives an extra request because both EJB1a and EJB1b have the same server weight value of 2, while EJB2 has a server weight value of 3.

*Example 6-5 J2EE client calling EJBs - server weighted round robin routing*

```
<Session Bean>
---> {beanServerName=EJB1a, beanServerProcessId=2376, beanNodeName=app1w
Execution time: 15
<=====>
```

```

<Session Bean>
----> {beanServerName=EJB1b, beanServerProcessId=2456, beanNodeName=applw
Execution time: 0
<=====>
<Session Bean>
----> {beanServerName=EJB2, beanServerProcessId=1936, beanNodeName=app2w}
Execution time: 0
<=====>
<Session Bean>
----> {beanServerName=EJB1a, beanServerProcessId=2376, beanNodeName=applw
Execution time: 0
<=====>
<Session Bean>
----> {beanServerName=EJB1b, beanServerProcessId=2456, beanNodeName=applw
Execution time: 15
<=====>
<Session Bean>
----> {beanServerName=EJB2, beanServerProcessId=1936, beanNodeName=app2w}
Execution time: 16
<=====>
<Session Bean>
----> {beanServerName=EJB2, beanServerProcessId=1936, beanNodeName=app2w}
Execution time: 0
<=====>
<Session Bean>
----> {beanServerName=EJB1a, beanServerProcessId=2376, beanNodeName=applw}
Execution time: 16
<=====>
<Session Bean>
----> {beanServerName=EJB1b, beanServerProcessId=2456, beanNodeName=applw}
Execution time: 94
<=====>
<Session Bean>
----> {beanServerName=EJB2, beanServerProcessId=1936, beanNodeName=app2w}
Execution time: 16
<=====>
<Session Bean>
----> {beanServerName=EJB1a, beanServerProcessId=2376, beanNodeName=applw}
Execution time: 15
<=====>
<Session Bean>
----> {beanServerName=EJB1b, beanServerProcessId=2456, beanNodeName=applw}
Execution time: 16
<=====>
<Session Bean>
----> {beanServerName=EJB2, beanServerProcessId=1936, beanNodeName=app2w}
Execution time: 0
<=====>

```

---

## 6.6.2 Prefer local

With the Prefer local policy, the selection made by the WLM plug-in not only depends on the running cluster members, but also on the system where the request comes from. The WLM plug-in will only select cluster members on the same system as the client, unless all local cluster members are unavailable.

This client might also be a Java virtual machine not running in WebSphere, such as a J2EE client application, or a stand-alone Java program accessing EJBs. For a J2EE client application, the Prefer local policy has no influence on the request distribution because the client runs on a remote workstation.

In the case of a Java program running on the same machine as WebSphere and using the WebSphere JRE and its ORB, the Prefer local policies will dispatch requests among EJB containers running on the same node. EJB WLM requires the WebSphere ORB and its WLM plug-in. If non-WebSphere ORBs are used, then the Java client will not be able to participate in EJB WLM.

**Important:** Whatever the server selection policy, process affinity and transaction affinity will always override the selection policy.

As you can see in Figure 6-12 on page 244, EJB requests are distributed across the available cluster members on the local node, app1w, in a repetitive order. Remember, both EJB1a and EJB1b have the same server weight value of 2.

Iteration	Servlet Node	Servlet Server Name	Servlet PID	Bean Node	Bean Server Name	Bean PID
1	app1w	WEBClient	2224	app1w	EJB1b	2324
2	app1w	WEBClient	2224	app1w	EJB1a	2088
3	app1w	WEBClient	2224	app1w	EJB1b	2324
4	app1w	WEBClient	2224	app1w	EJB1a	2088
5	app1w	WEBClient	2224	app1w	EJB1b	2324
6	app1w	WEBClient	2224	app1w	EJB1a	2088
7	app1w	WEBClient	2224	app1w	EJB1b	2324
8	app1w	WEBClient	2224	app1w	EJB1a	2088
9	app1w	WEBClient	2224	app1w	EJB1b	2324
10	app1w	WEBClient	2224	app1w	EJB1a	2088

Figure 6-12 Web client calling EJBs - Prefer local policy enabled

The output listing in Example 6-6 shows a similar behavior for our J2EE client application, which is run from node app1w. EJB requests are distributed over the available cluster members on the local node, app1w, in a repetitive order because both EJB1a and EJB1b have the same server weight value of 2.

Example 6-6 J2EE client calling EJBs - Prefer local policy enabled

```
<Session Bean>
--> {beanServerName=EJB1b, beanServerProcessId=2324, beanNodeName=app1w}
Execution time: 31
<=====>
<Session Bean>
--> {beanServerName=EJB1a, beanServerProcessId=2088, beanNodeName=app1w}
Execution time: 0
<=====>
<Session Bean>
```

```

---> {beanServerName=EJB1b, beanServerProcessId=2324, beanNodeName=app1w}
Execution time: 0
<=====>
<Session Bean>
---> {beanServerName=EJB1a, beanServerProcessId=2088, beanNodeName=app1w}
Execution time: 15
<=====>
<Session Bean>
---> {beanServerName=EJB1b, beanServerProcessId=2324, beanNodeName=app1w}
Execution time: 16
<=====>
<Session Bean>
---> {beanServerName=EJB1a, beanServerProcessId=2088, beanNodeName=app1w}
Execution time: 0
<=====>
<Session Bean>
---> {beanServerName=EJB1b, beanServerProcessId=2324, beanNodeName=app1w}
Execution time: 0
<=====>
<Session Bean>
---> {beanServerName=EJB1a, beanServerProcessId=2088, beanNodeName=app1w}
Execution time: 16
<=====>
<Session Bean>
---> {beanServerName=EJB1b, beanServerProcessId=2324, beanNodeName=app1w}
Execution time: 0
<=====>
<Session Bean>
---> {beanServerName=EJB1a, beanServerProcessId=2088, beanNodeName=app1w}
Execution time: 0
<=====>
<Session Bean>
---> {beanServerName=EJB1b, beanServerProcessId=2324, beanNodeName=app1w}
Execution time: 0
<=====>
<Session Bean>
---> {beanServerName=EJB1a, beanServerProcessId=2088, beanNodeName=app1w}
Execution time: 0
<=====>
<Session Bean>
---> {beanServerName=EJB1b, beanServerProcessId=2324, beanNodeName=app1w}
Execution time: 0
<=====>

```

---

### 6.6.3 Process affinity

Whatever the workload management server selection policy, if an EJB is available in the same cluster member as the client, all the requests coming from the client will be directed to this EJB. This is called *process affinity*, because all the requests are in-process requests. The advantage of process affinity is that there is no need for serialization for method calls. Parameters can be passed by value without any serialization costs, since all method calls are performed within the same Java virtual machine, in the same memory space.

To take advantage of process affinity, the client can only be a servlet or an EJB. In the case of a servlet, process affinity is only possible if the Web container running the servlet is in the same application server as the EJB container. In the case of an EJB (a stateless session bean acting as a facade, for instance, as recommended by EJB development best practices), process affinity occurs when the called EJB is in the same EJB container as the calling EJB. With IBM WebSphere Application Server Network Deployment V5.1, you can only have one EJB container per application server.

### 6.6.4 Transaction affinity

When several requests to EJB methods occur in the scope of the same transaction (a user-managed transaction or a container-managed transaction), all requests will go to the same cluster member, if possible. As soon as the WLM plug-in notices that a transaction is started, it will stop dispatching the requests to different cluster members. All requests within the scope of this transaction are sent to the same cluster member.

## 6.7 EJB workload management failover

This section is a short summary on EJB failover. EJB workload management failover is covered in more detail in 9.4, “EJB container failover” on page 365.

The EJB workload management plug-in uses the following failover strategies:

- ▶ If the workload management plug-in cannot contact a cluster member, it automatically redirects the request to another cluster member, providing automatic failover.
- ▶ If the application throws an exception, automatic failover does not occur. The workload management plug-in does not retry the request because it cannot know whether the request was completed.
- ▶ The workload management plug-in will catch and wrap communication failure exceptions in a `TRANSIENT` exception in those cases where transparent

failover could not be achieved because it is unknown exactly how much processing the server did before the failure occurred. But should the application execute some compensation logic to get the state in a server back to where it should be (by performing a rollback or whatever needs to be done), then the request could be re-executed with potentially a different result (success). So, while the request could not transparently be failed over, there are still other cluster members available and if the request were tried again it might succeed. The application compensation and retry logic should stop retrying when a `NO_IMPLEMENT` exception is thrown instead of a `TRANSIENT` exception, because this indicates that no servers are available to handle the request.

This means that there are two different ways of how the EJB workload management plug-in handles failure situations:

- ▶ Exceptions that workload management reacts to by transparently sending the failed request to a different server that may complete it successfully. These exceptions are listed in 6.7.1, “Exceptions triggering automatic failover” on page 247.
- ▶ Exceptions that WLM throws out to the application to react to. These exceptions are listed in 6.7.2, “Exceptions thrown by WLM to the application” on page 248.

### 6.7.1 Exceptions triggering automatic failover

The following is a list of exceptions that the EJB workload management will react to. Depending on the exception, it will redirect the failed request to another cluster server:

- ▶ If the root exception `org.omg.CORBA.COMM_FAILURE` or the `org.omg.CORBA.NO_RESPONSE` exception is thrown, their return value will determine whether automatic failover occurs:
  - With a `COMPLETION_STATUS` of `COMPLETED_NO`, automatic failover occurs because the request was not completed.
  - With a `COMPLETION_STATUS` of `COMPLETED_YES`, failover does not occur because the request was successfully completed.
  - With a `COMPLETION_STATUS` of `COMPLETED_MAYBE`, automatic failover does not occur. The workload management plug-in cannot verify whether the request was completed. In this situation, the client application must anticipate a failure and retry the request. The workload management plug-in then attempts to direct the request to a surviving cluster member.

- ▶ `org.omg.CORBA.COMM_FAILURE`  
This exception is thrown by the ORB when a communications failure occurs. Any current transactions are rolled back and non-transactional requests are redone.
- ▶ `org.omg.CORBA.NO_RESPONSE`  
This exception is thrown by the ORB when a communication failure occurs after a connection has been established, for example because a timeout occurs before getting a response.
- ▶ `org.omg.CORBA.TRANSCIENT`  
This exception is thrown by the ORB when a connection could not be established when the application tried to invoke a request.

### 6.7.2 Exceptions thrown by WLM to the application

The following is a list of exceptions that the EJB workload management throws out to the application to react to:

- ▶ If a `org.omg.CORBA.TRANSCIENT` with minor code: 1229066304 (0x49421040) exception is thrown, the workload management plug-in found that the cluster member was in server quiesce mode.
- ▶ If a `org.omg.CORBA.TRANSCIENT` with minor code: 1229066306 (0x49421042) exception is thrown, the workload management plug-in had a connection problem with a server.
- ▶ If a `com.ibm.CORBA.INTERNAL` with minor code: 1229066304 (0x49421040) exception is thrown, the workload management plug-in is no longer operating properly and no failover occurs.
- ▶ If a `com.ibm.CORBA.NO_IMPLEMENT` with minor code: 1229066304 (0x49421040) exception is thrown, the workload management plug-in has attempted repeatedly to contact all the servers without success. Workload management resumes when servers become available again.

## 6.8 Backup Cluster support

**Restriction:** This function is only available in IBM WebSphere Application Server Enterprise V5.0.2 and V5.1. It is *not* available in IBM WebSphere Application Server Network Deployment.

IBM WebSphere Application Server Enterprise V5.0.2 and V5.1 supports a mirrored backup cluster that can fail over IIOP requests from a primary cluster in



one cell into the mirrored backup cluster in another cell if the primary cluster should fail.

Before you can enable this backup cluster support, you need to create the backup cluster in another cell with the same cluster name as the primary cluster and deploy the same applications into both the primary cluster and the mirrored backup cluster, with the same resources in the backup cluster as in the primary cluster. The primary cluster and the backup cluster must reside in separate cells. The backup cluster bootstrap host and port determine which cell contains the backup cluster.

Then you need to enable backup cluster support in the Administrative Console:

1. Select **Servers -> Clusters -> <cluster\_name>**. From the Additional Properties pane of the Configuration tab, select **Backup Cluster**. See Figure 6-13.

**SPECjCluster**

A server cluster consists of a group of application servers. If one of the member servers fails, requests will be routed of the cluster. [i](#)

**Runtime** **Configuration** **Local Topology**

**General Properties**

Cluster name:	* SPECjCluster	<a href="#">i</a> The name of this cluste
Prefer local	<input type="checkbox"/>	<a href="#">i</a> Specifies whether ente requests will be routed to 1 which the client resides w

Apply OK Reset Cancel

**Additional Properties**

<a href="#">Cluster members</a>	An application server that belongs to a group of servers called a cluster.
<a href="#">Backup Cluster</a>	Specifies the cluster designated as the backup.
<a href="#">Weight Advisor</a>	Specifies the source or generator of load balancing weights.

Figure 6-13 Backup Cluster configuration

2. Select **Domain Bootstrap Address**, again from the Additional Properties pane. This launches a window (shown in Figure 6-14 on page 250) where you need to enter the **Host** and **Port** of the foreign cell's Deployment Manager.

As you can see, you have two choices for the setting: Runtime and Configuration. The difference is that the runtime setting will not survive a Deployment Manager restarts, so it is temporary while the configuration setting remains active after a Deployment Manager restart.

[Server Cluster](#) >

### Domain Bootstrap Address

The bootstrap host and port for the deployment manager that contains the backup cluster. [i](#)

**Runtime** **Configuration**

**General Properties**

Host	<input type="text" value="haowang.itso.ibm.com"/>	<a href="#">i</a> The bootstrap host for the deployment manager of the backup cluster.
Port	<input type="text" value="9809"/>	<a href="#">i</a> The bootstrap port for the deployment manager of the backup cluster.

Apply OK Reset Cancel

Figure 6-14 Backup cluster support - Enter bootstrap Host and Port

- If you do not know the host and port, you can find this information using the Administrative Console on the foreign cell's Deployment Manager. Select **System Administration -> Deployment Manager -> End Points -> BOOTSTRAP\_ADDRESS**. The host and port are displayed as shown in Figure 6-15.

[dmgr](#) > [End Points](#) >

### BOOTSTRAP\_ADDRESS

Configure important TCP/IP ports which this server uses for connections. [i](#)

**Configuration**

**General Properties**

End Point Name	BOOTSTRAP_ADDRESS <a href="#">i</a>	
Host	* <input type="text" value="haowang.itso.ibm.com"/>	<a href="#">i</a> The IP address, DNS host name with domain name suffix, or just the DNS host name, used by a client to request a Web application resource (such as a servlet, JSP, or HTML page).
Port	* <input type="text" value="9809"/>	<a href="#">i</a> The port for which the Web server has been configured to accept client requests. Specify a port value in conjunction with the host name.

Apply OK Reset Cancel

Figure 6-15 Find the bootstrap address of the foreign cell

- Once you have found the bootstrap host and port, enter this information into the backup cluster configuration page shown in Figure 6-14, click **OK** and **Save** the configuration.

5. If you performed the previous steps on the Configuration tab, then restart the Deployment Manager for this change to take effect. If you changed the settings on the Runtime tab, then these changes take effect immediately.

This is all that needs to be done to enable the backup cluster. If after a failover cluster members in the primary cluster subsequently become available again, the mirrored cluster support attempts to fail back the requests to the primary cluster. This fail back is automatic.

For mutual failback support the primary cluster must be defined as a backup for the backup cluster. In other words, both primary and backup clusters must have a backup configured, and each cluster's backup must point to the opposite cluster.

Please note that the mirrored cluster failover support is not a cell level failover or Node Agent failover. For the most part, the mirrored cluster support depends on a running Deployment Manager and Node Agents, and is limited to cluster failover only. For example, if the primary cluster's entire cell stops processing, a new client's requests to the primary cluster will fail and is not sent to the backup cluster. This is because information regarding the backup cluster cannot be retrieved from the primary cluster's cell because the primary cluster's cell is not processing. On the other hand, if the primary cluster's cell Deployment Manager or Node Agents and application servers stop processing *after* the client has already been sending requests to the primary cluster, the client already knows about the backup cluster and is able to send the requests to the backup cluster.

For more information about this topic, see the WebSphere InfoCenter.





## Part 3

# Implementing the solution





## Implementing the sample topology

This chapter provides instructions on how to implement our sample topology. This sample topology is used to demonstrate the WebSphere Application Server V5.1 scalability and high availability features.

We also describe how to install and configure the J2EE applications BeenThere and Trade3.1.

The chapter contains the following sections:

- ▶ “Installation summary” on page 260
- ▶ “Configuring Caching Proxy” on page 260
- ▶ “Configuring Load Balancer” on page 264
- ▶ “Configuring WebSphere clusters” on page 276
- ▶ “Installing and configuring BeenThere” on page 290
- ▶ “Installing and configuring Trade3.1” on page 298

**Tip:** You can go directly to the various sections of this chapter if you do not plan to implement the entire scenario. For example, go to 7.5, “Configuring WebSphere clusters” on page 276 if you do not plan to implement the Caching Proxy and/or Load Balancer but only want to use a cluster and BeenThere or Trade3.1.

## 7.1 Overview

This section gives you a quick overview of our sample topology. This includes the software needed, a description of our sample topology, and the applications installed in our test environment.

**Important:** This chapter covers both the AIX platform and the Windows 2000 platform. We describe the implementation of the sample topology on the Windows 2000 platform. However, most steps are exactly the same on both platforms. If there are platform differences, we will emphasize this.

### 7.1.1 Software products

We are using Windows 2000 and AIX 5.2 in our test environment. The following versions of these operating systems have been used:

- ▶ Microsoft Windows 2000 Server with Service Pack 4
- ▶ IBM AIX 5.2 Maintenance Level 2 with 32 bit (unix\_up) kernel

The following IBM software is used for the WebSphere implementation:

- ▶ IBM WebSphere Edge Components V5.1
- ▶ IBM JDK 1.4.1
- ▶ IBM HTTP Server 2.0.42.2
- ▶ WebSphere Application Server V5.1
- ▶ IBM WebSphere Application Server Network Deployment V5.1
- ▶ IBM DB2 UDB V8.1 with Fix Pack 3, 32 bit version

More information about the minimum hardware and software requirements for WebSphere and DB2 UDB can be found at:

<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>  
<http://www.ibm.com/software/data/db2/udb/sysreqs.html>

### 7.1.2 The sample topology

Our sample topology on the Windows 2000/AIX platform demonstrates the following key features:

- ▶ Web server caching, workload management, horizontally (for HTTP requests)
- ▶ Web container workload management, both horizontally and vertically (for handling servlet requests)



- ▶ EJB container workload management, both horizontally and vertically

Figure 7-1 on page 258 explains the sample topology:

- ▶ A cluster of two Web servers hosts cluster.itso.ibm.com running the Load Balancer function, anticipated by a Caching Proxy, both from WebSphere Edge Components.
- ▶ A dedicated Deployment Manager machine managing the WebSphere Application Server cell, running IBM WebSphere Application Server Network Deployment V5.1.
- ▶ Within this cell, we have a WebSphere cluster with various application server processes. WebSphere Application Server V5.1 is installed on both machines.
- ▶ A dedicated database server running IBM DB2 UDB V8.1.

**Important:** While this chapter describes how to configure a split JVM topology with the Web containers and EJB containers running in separate application servers, this is done so only for informational and demonstration purposes.

Recall the note from 3.3, “Strategies for scalability” on page 56 that a negative performance impact will likely result when an application is deployed in this fashion. Better performance, scalability and failover is achieved when the Web components for an application are deployed in the same application server as the EJB components.

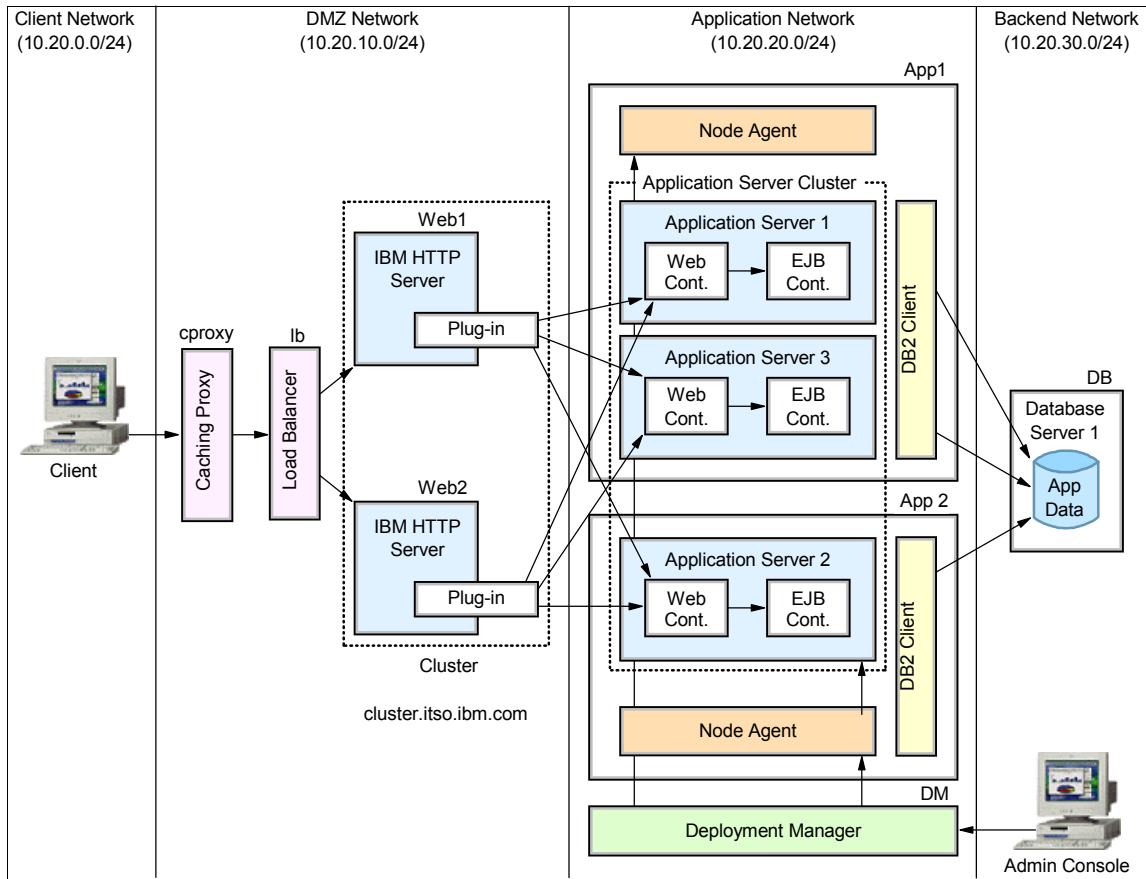


Figure 7-1 The sample topology

Table 7-1 shows the names, functions, locations, and IP addresses of our systems. All machines are part of the same network. However, we choose different IP address ranges to distinguish between the DMZ network, the application network, and the back-end network.

Table 7-1 Machines in the sample topology

Name	Network	IP address	Subnet mask	Function
cproxy	DMZ Network	10.20.10.101	255.255.0.0	Caching Proxy
lb	DMZ Network	10.20.10.102	255.255.0.0	Load Balancer
web1	DMZ Network	10.20.10.103	255.255.0.0	Web Server 1
web2	DMZ Network	10.20.10.104	255.255.0.0	Web Server 2

Name	Network	IP address	Subnet mask	Function
dm	Application Network	10.20.20.100	255.255.0.0	Deployment Manager
app1	Application Network	10.20.20.103	255.255.0.0	Application servers system 1
app2	Application Network	10.20.20.104	255.255.0.0	Application servers system 2
db	Backend Network	10.20.30.100	255.255.0.0	Database server

Next Table 7-2, shows the properties of our Web server cluster.

*Table 7-2 Cluster in the sample topology*

Name	Network	IP address	Subnet mask	Function
cluster	DMZ Network	10.20.10.100	255.0.0.0	Web server cluster

### 7.1.3 Applications used in our sample topology

We have used two sample applications in our environment.

#### **BeenThere**

BeenThere is a simple, lightweight J2EE application. It is very useful for demonstrating workload management as it shows you which application server responded to a request. For more information about BeenThere see “Installing and configuring BeenThere” on page 290.

#### **Trade3.1**

Trade3.1 is the third generation of the WebSphere end-to-end benchmark and performance sample application. The new Trade3.1 benchmark has been re-designed and developed to cover WebSphere's significantly expanding programming model. This provides a real-world workload driving WebSphere Application Server V5.1 implementation of J2EE 1.3 and Web Services including key WebSphere performance components and features.

Trade3.1's new design spans J2EE 1.3 including the new EJB 2.0 component architecture, message-driven beans, transactions (one-phase, two-phase commit) and Web services (SOAP, WSDL, UDDI). Trade3.1 also highlights key WebSphere performance components such as dynamic cache and WebSphere Edge Server. For more information about Trade 3.1 see “Installing and configuring Trade3.1” on page 298.

## 7.2 Installation summary

Table 7-3 summarizes the installation procedure. These steps need to be carried out *before* proceeding with the rest of this chapter. We do not describe how to install each individual component but explain how to configure the already installed software.

**Important:** Throughout this chapter we are using the default application install directories. If you are *not* using these defaults, make sure to make the proper changes to the instructions provided here.

**Tip:** Instead of accepting the default installation directory for WebSphere Application Server V5.1 on the Windows 2000 platform (C:\Program Files\WebSphere\AppServer) we suggest to install WebSphere in a path that does not have any spaces or other special characters. For example C:\WebSphere\AppServer.

Table 7-3 Installation summary

Step number	Action	System
1	Install IBM DB2 UDB V8.1 FP3	db
2	Install IBM DB2 V8.1 FP3 Administrative client and Runtime client	app1, app2
3	Catalog DB2 node on clients	app1, app2
4	Install IBM WebSphere Application Server V5.1	app1, app2
5	Install IBM WebSphere Application Server Network Deployment V5.1	dm
6	Install IBM HTTP Server 2.0.42 and WebSphere Application Server V5.1 plug-in. IBM JDK 1.4.1 might be necessary.	web1, web2
7	Install IBM WebSphere Edge Components 5.1	cproxy, lb

## 7.3 Configuring Caching Proxy

WebSphere Edge Components V5.1 contains a Caching Proxy function which can cache often requested documents, cutting down in that way the response time.

### 7.3.1 Starting Caching Proxy

To start Caching Proxy in Windows 2000 (if not started automatically), select **Start -> Settings -> Control Panel -> Administrative Tools -> Services**. Right-click **IBM Caching Proxy** and select **Start**. (To stop the service, follow the same steps and select **Stop**.)

**AIX:** On the AIX platform, start the Caching Proxy by running the following command:

```
startsrc -s ibmproxy
```

To stop the service type:

```
stopsrc -s ibmproxy
```

### 7.3.2 Set up Caching Proxy

To configure Caching Proxy, you need to edit the Caching Proxy configuration file in directory `C:\Program Files\IBM\edge\cp\etc\en_US\ibmproxy.conf`:

1. Open the configuration file using WordPad (or NotePad).

**AIX:** On the AIX platform, the file `ibmproxy.conf` is located in directory `opt/ibm/edge/cp/etc/en_US`. Open the file using the vi editor, for example.

2. Search for the Hostname directive in the configuration file. Change the default value to the **fully qualified name of your host**, as illustrated in Figure 7-2.

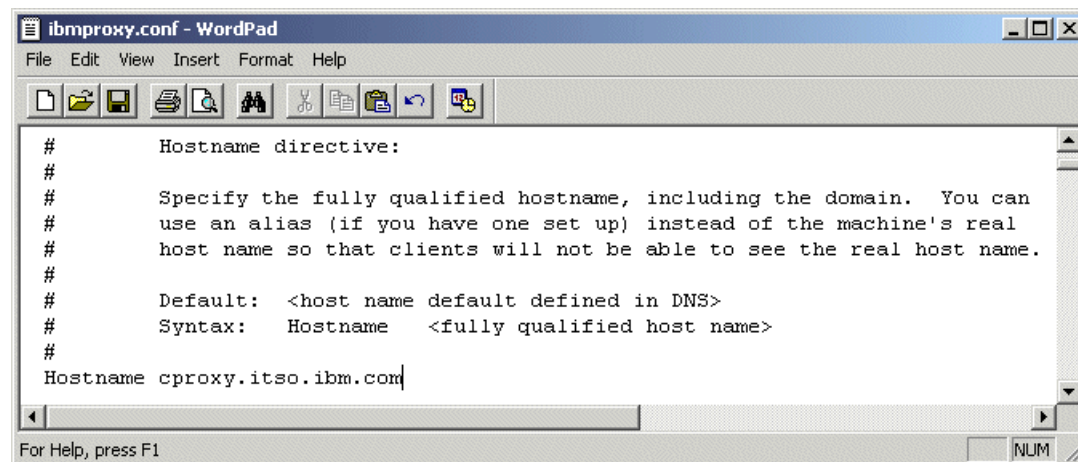


Figure 7-2 Caching Proxy configuration - set fully qualified hostname

3. Look for the Port directive. Its default value is 80. We accept that for now. If you run another service on port 80, for example an HTTP server, then change it to another value. The most common port number for a proxy server is 8080.
4. If you have lots of traffic on your system, you should move the log files to a different drive or volume group. To do that, search for *AccessLog* which is found in the Logging directives section. Here you will find also other log file directives. Change them according to your needs.
5. As the last but most important configuration step look for the following text:

```
*** START NEW MAPPING RULES SECTION ***
```

Below this line, you need to insert three new lines as shown in Example 7-1 (Proxy, ReversePass for http, and ReversePass for https). Please note that the text for these lines needs to be on one line!

#### Example 7-1 Proxy configuration directives

---

```
# NOTE: The installation defaults should be added below
# *** START NEW MAPPING RULES SECTION ***
Proxy      /* http://cluster.itso.ibm.com/*      cproxy.itso.ibm.com
ReversePass http://cluster.itso.ibm.com/* http://cproxy.itso.ibm.com/* cproxy.itso.ibm.com
ReversePass https://cluster.itso.ibm.com/* https://cproxy.itso.ibm.com/* cproxy.itso.ibm.com

# *** END NEW MAPPING RULES SECTION ***
```

---

6. Save the configuration file.

### 7.3.3 Set administrator user ID and password

To be able to use the Web browser based administration interface, you need to set a user ID and a password as follows:

1. Open a Windows Command Prompt window and issue the following command (please note that the whole command must be typed on one line):

```
htadm -adduser "c:\Program
Files\IBM\edge\cp\server_root\protect\webadmin.passwd"
```

**AIX:** On the AIX platform, run the following command from a terminal window:

```
# htadm -adduser /opt/ibm/edge/cp/server_root/protect/webadmin.passwd
```

2. When asked for by the program, provide a user ID, a password, and a real user name for the Web administrator user.

### 7.3.4 Restart Caching Proxy

After the configuration is complete, you have to restart the Proxy server:

1. Open a Browser window and go to the Web page of the Caching Proxy. Type the following address in your browsers' URL field:

`http://<your_hostname>/`

2. The Caching Proxy Front Page appears. Click **CONFIGURATION AND ADMINISTRATION FORMS**.
3. Log on with the administrator user ID and password previously created.

**Note:** You might need to authenticate twice until you reach the administrator window.

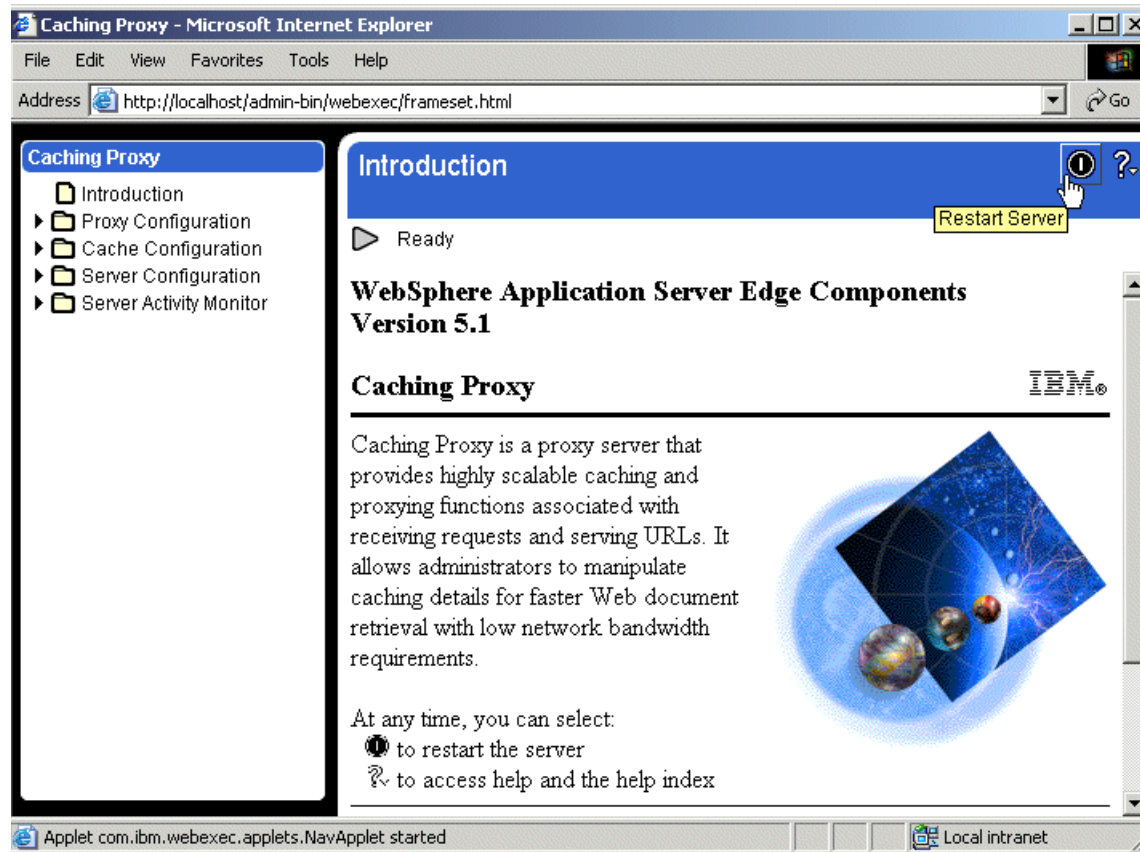


Figure 7-3 Restart Caching Proxy

4. Click the **Restart** button in the right-top corner as illustrated in Figure 7-3 on page 263. As shown in Figure 7-4, a message will indicate that the restart was successful.

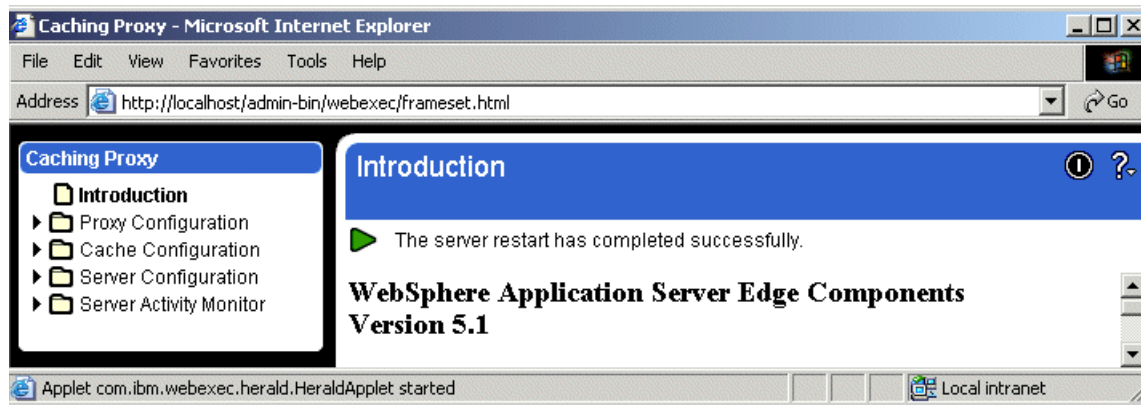


Figure 7-4 Caching Proxy restarted successfully

## 7.4 Configuring Load Balancer

The Load Balancer function from the WebSphere Edge Components V5.1 allows you to implement load balancing for HTTP requests among two or more Web servers. In our sample topology we defined a cluster named cluster.itso.ibm.com with IP address 10.20.10.100. This is illustrated in Figure 7-5.

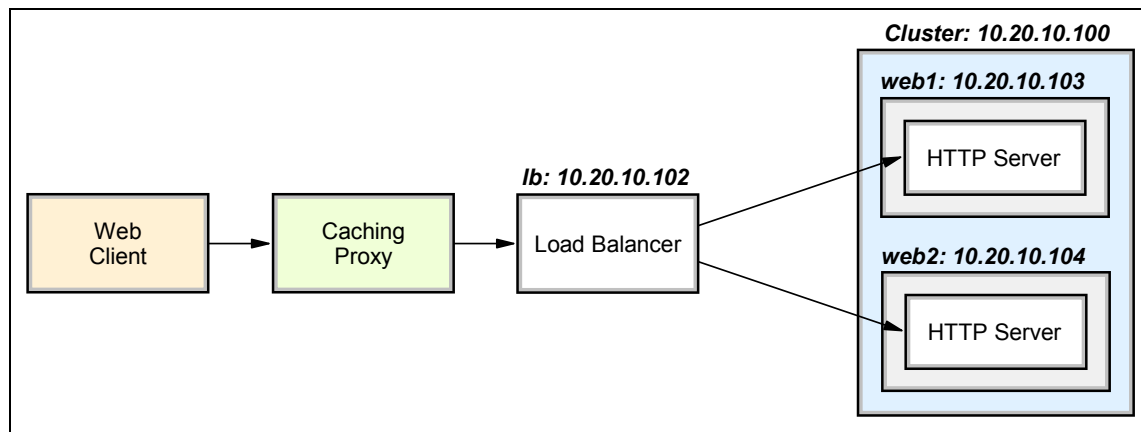


Figure 7-5 Load Balancer and Web server topology



## 7.4.1 Configuring the Web servers for Load Balancer

Before configuring Load Balancer, the Web servers need a loopback alias that points to the cluster address of cluster.itso.ibm.com (10.20.10.100). To configure this, follow these steps for each Web server on Windows 2000:

1. Log on as a user with administrator rights and click **Start -> Settings -> Control Panel**.
2. To add the MS Loopback Adapter Driver:
  - a. Double-click **Add/Remove Hardware**. This launches the Add/Remove Hardware Wizard.
  - b. Click **Next**, select **Add/Troubleshoot a Device**, then click **Next**. The Choose a Hardware Device window is displayed.
  - c. Scroll down to check if the MS Loopback Adapter is already in the list of installed hardware. If so, you can click **Cancel** to exit.
  - d. If the MS Loopback Adapter is not in the list, select **Add a New Device** and click **Next**.
  - e. Do *not* search for new hardware (the default). Instead, choose the option to select new hardware from a list and click **Next**.
  - f. Select **Network Adapters** and click **Next**.
  - g. On the Select Network Adapter window, select **Microsoft** from the Manufacturers list and then **Microsoft Loopback Adapter** from the Network Adapter list. Click **Next** twice to install the Loopback Adapter.
  - h. Click **Finish** to complete the installation.
3. From the Control Panel, double-click **Network and Dial-up Connections**.
4. Select the connection that is using the Microsoft Loopback Adapter and right-click it. Select **Properties** from the context menu.
5. Select **Internet Protocol (TCP/IP)**, then click **Properties**.
6. Check **Use the following IP address**. For the IP address enter the cluster address of **10.20.10.100**. For the Subnet mask enter **255.0.0.0**.

### Notes:

- Please note that the Subnet mask for the cluster address is 255.0.0.0 and not 255.255.0.0.
- Don't enter a gateway address. Use the localhost as the default DNS server.

**AIX:** To add the loopback alias in AIX, use the following command:

```
ifconfig lo0 alias cluster.itso.ibm.com netmask 255.255.0.0
```

For other operating systems, refer to the *Load Balancer Administration Guide*, GC09-4602.

## 7.4.2 Starting Load Balancer and administration GUI

To start Load Balancer in Windows 2000, select **Start -> Settings -> Control Panel -> Administrative Tools -> Services**. Right-click **IBM Dispatcher** and select **Start**. (To stop the service, follow the same steps and select **Stop**.)

To start the Load Balancer administration GUI tool, click **Start -> Programs -> IBM WebSphere -> Edge Components -> Load Balancer -> Load Balancer**.

**AIX:** To start Load Balancer under AIX, run the following command as *root* user:

```
/usr/bin/dsserver
```

To start the Load Balancer administration GUI tool, run the following command:

```
/usr/bin/lbadmin
```

**Note:** These commands can be used on other platforms as well, for example on the Windows 2000 platform. Just consider the correct path.

## 7.4.3 Connecting to the Dispatcher host

Once the administration GUI is started, you need to connect to the Dispatcher host. To connect to the host:

1. Right-click **Dispatcher**, as shown in Figure 7-6 on page 267, and click **Connect to Host....**

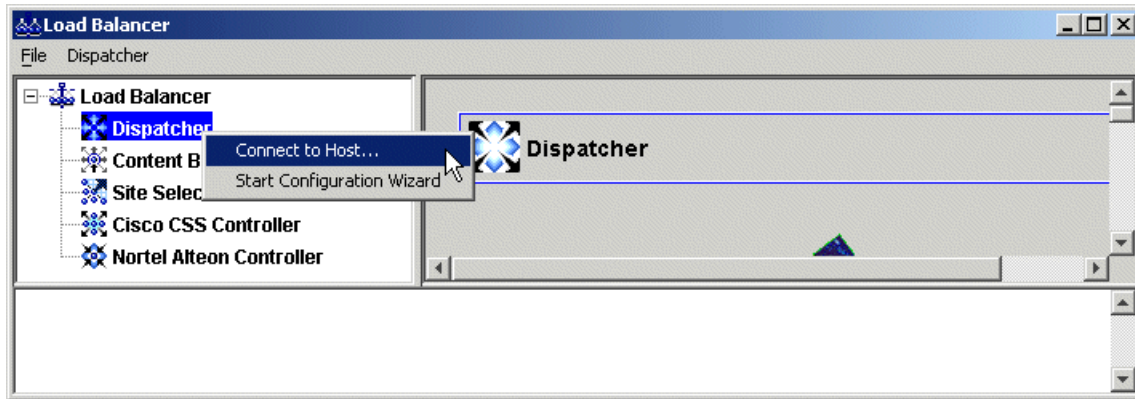


Figure 7-6 Connect to Dispatcher host

2. The Dispatcher Login window appears as shown in Figure 7-7. Accept the default hostname and click **OK**.

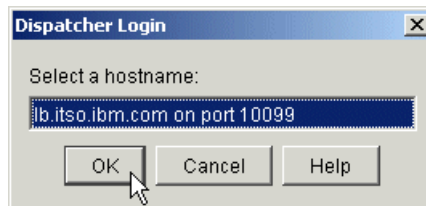


Figure 7-7 Dispatcher login window

#### 7.4.4 Adding the Web server cluster

The next step is to add the Web server cluster to the Load Balancer configuration. In our case the cluster name is cluster.itso.ibm.com and the IP address is 10.20.10.100.

**Note:** If your hosts are not defined in an DNS server, you need to add the host names and IP addresses to the C:\winnt\system32\drivers\etc\hosts file of each system. For our environment, we added the entries listed in Example 7-2 to our lb, web1, and web2 hosts.

*Example 7-2 /etc/hosts sample*

```
...
10.20.10.100    cluster.itso.ibm.com    cluster
10.20.10.101    cproxy.itso.ibm.com    cproxy
10.20.10.102    lb.itso.ibm.com         lb
10.20.10.103    web1.itso.ibm.com       web1
10.20.10.104    web2.itso.ibm.com       web2
...
```

**AIX:** On the AIX platform, the file location is /etc/hosts.

To add a new cluster:

1. Start the Executor by right-clicking **Host: lb** on the Load Balancer administration GUI and select **Start Executor** from the pop-up menu, as shown in Figure 7-8.

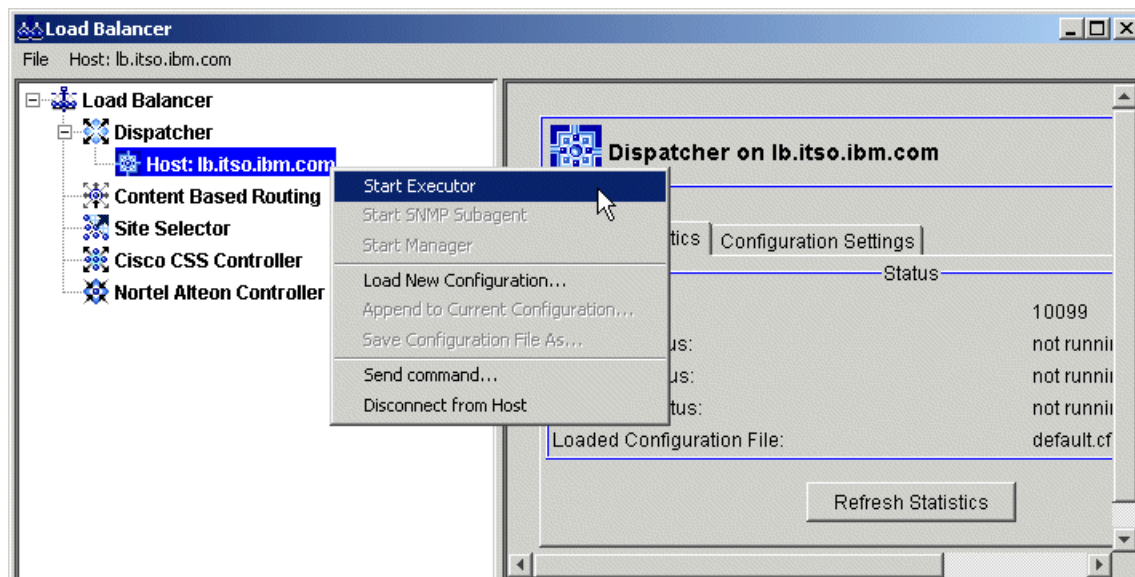


Figure 7-8 Starting the Executor

2. After starting the Executor, you need to add the cluster to the configuration. This is done by right-clicking **Executor** and selecting **Add Cluster...**, as shown in Figure 7-9.

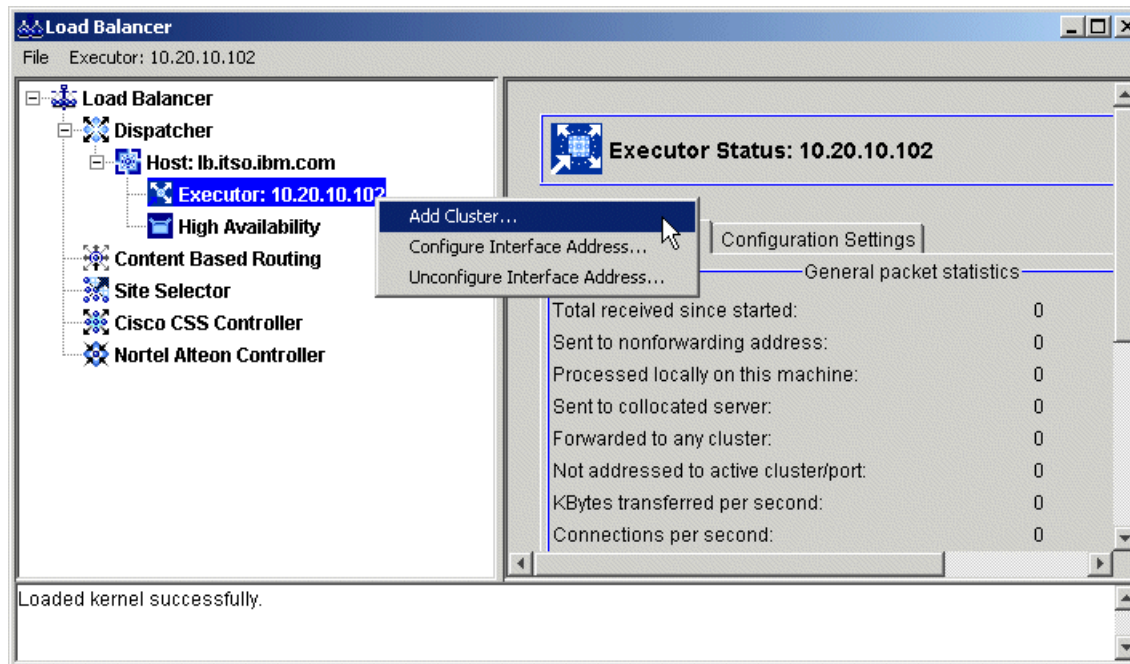


Figure 7-9 Adding the cluster

3. On the Add a cluster window enter an arbitrary name for the cluster. As mentioned before, we are using cluster.itso.ibm.com.
4. Enter the appropriate **IP address** in the Cluster address field, in our case 10.20.10.100.
5. Select the **Configure this cluster** option, as shown in Figure 7-10 on page 270, then click **OK**.

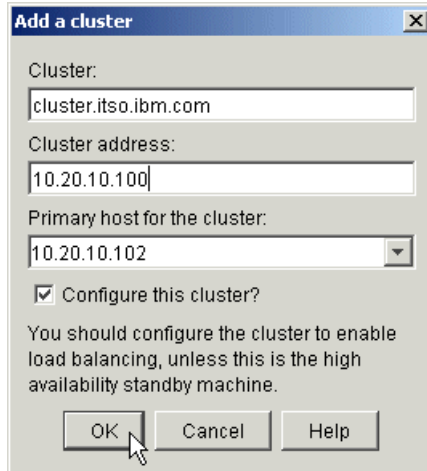


Figure 7-10 Adding a cluster window

6. Click **OK** and accept the defaults when the Configure cluster address window appears.

### 7.4.5 Adding a port to the cluster

The next step is to configure the port that will be used by the clients to access the WebSphere applications. In our case, this is port 80. To add the Port:

1. Right-click **Cluster: cluster.itso.ibm.com** and select **Add Port...** from the pop-up menu, as shown in Figure 7-11 on page 271.

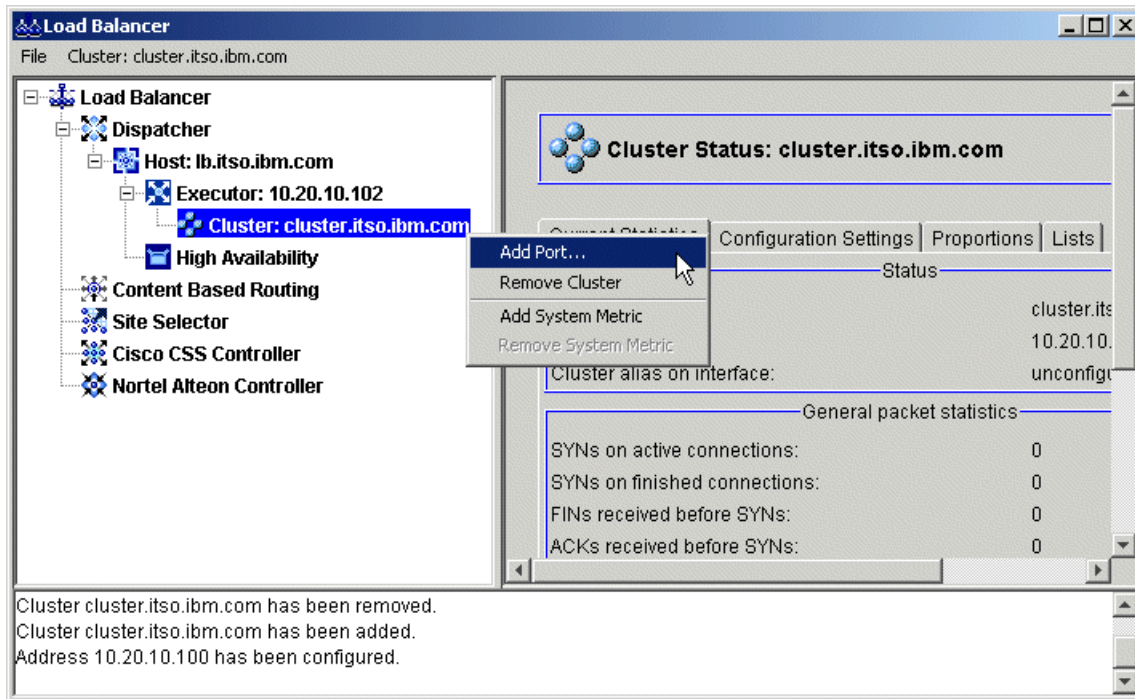


Figure 7-11 Adding a port

- When the Add a port window appears, enter **80** in the Port number field as shown in Figure 7-12, then click **OK**.

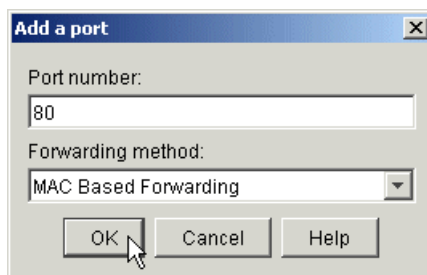


Figure 7-12 Specifying the port number

## 7.4.6 Adding the Web servers to the cluster

The next step is to add the Web servers to the cluster. Our Web servers are:

- ▶ web1.itso.ibm.com (web1), IP address 10.20.10.103
- ▶ web2.itso.ibm.com (web2), IP address 10.20.10.104

To add the Web servers to the cluster:

1. Right-click **Port: 80** and select **Add Server...**, as shown in Figure 7-13.

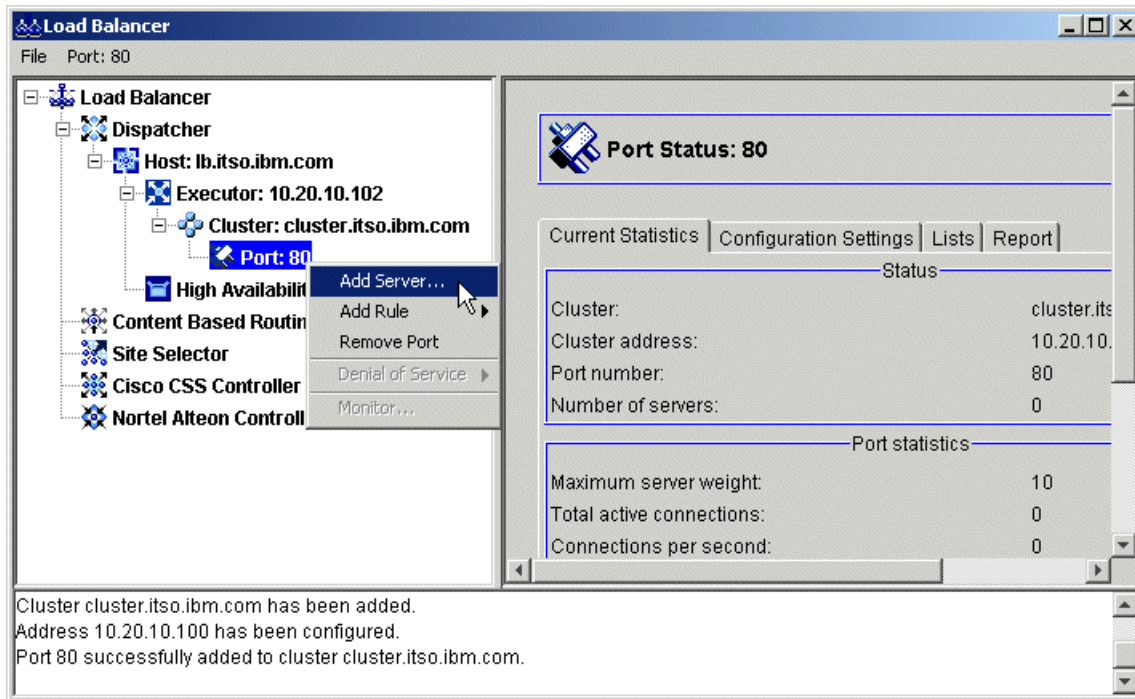


Figure 7-13 Adding a server

2. On the Add a server window (see Figure 7-14) enter **web1** into the Server field and the IP address **10.20.10.103** into the Server address field, then click **OK**.

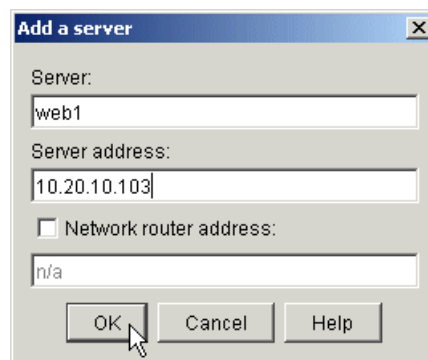


Figure 7-14 Adding the sever web1



3. Do the same for the second Web server, **web2** with IP address **10.20.10.104**.

The Load Balancer administration GUI now shows the two Web servers belonging to the cluster. See Figure 7-15.

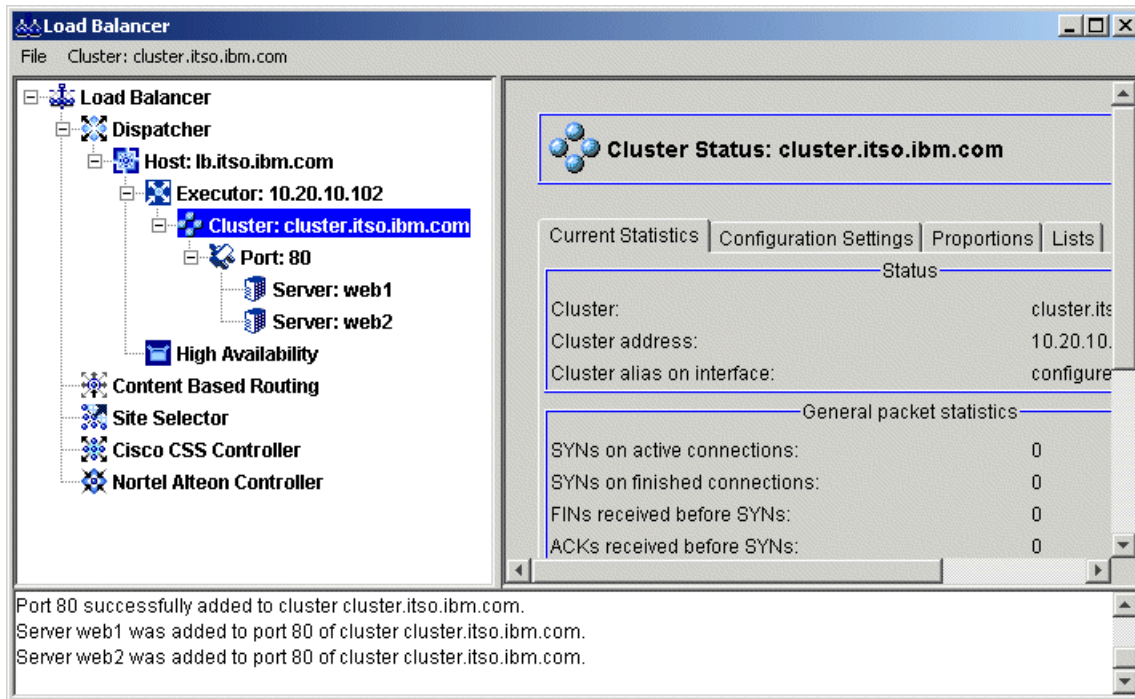


Figure 7-15 Configured cluster view

### 7.4.7 Start the Network Dispatcher manager

The Network Dispatcher manager component provides dynamic load balancing. To start the Network Dispatcher manager:

1. Right-click **Host: lb** and select **Start Manager** from the pop-up menu as shown in Figure 7-16 on page 274.

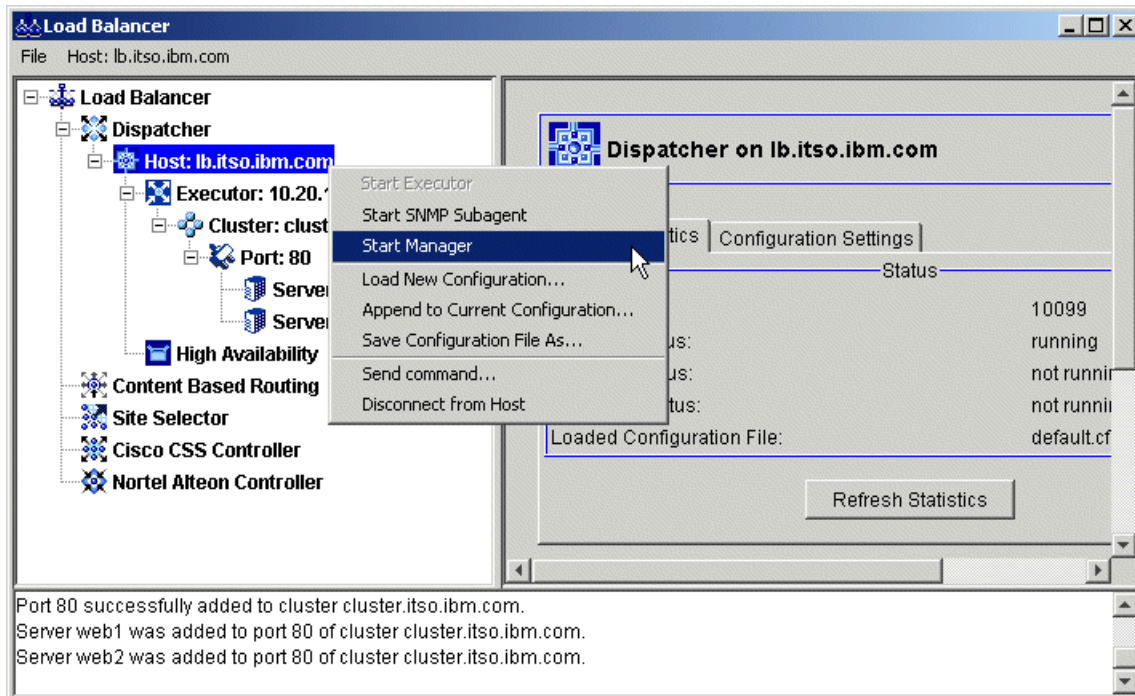


Figure 7-16 Starting the manager

2. When the Start the manager window appears as shown in Figure 7-17, accept the defaults and click **OK**.

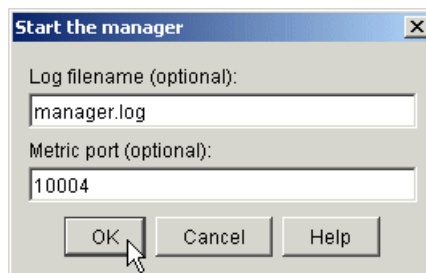


Figure 7-17 Start the manager default settings

3. When the Start an advisor window appears as shown in Figure 7-18 on page 275, accept the defaults and click **OK**.

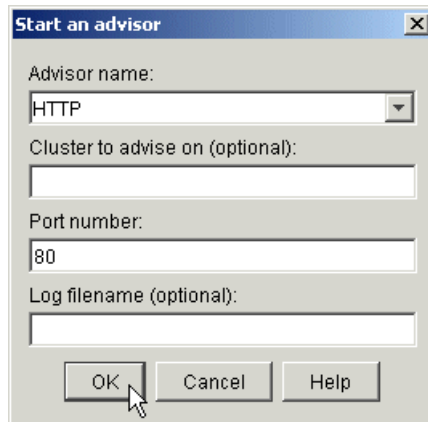


Figure 7-18 Advisor settings

4. Once the manager has been started, you see the Manager folder under your Host. Double-click **Manager** to see the advisor if it is not displayed automatically (see Figure 7-19).

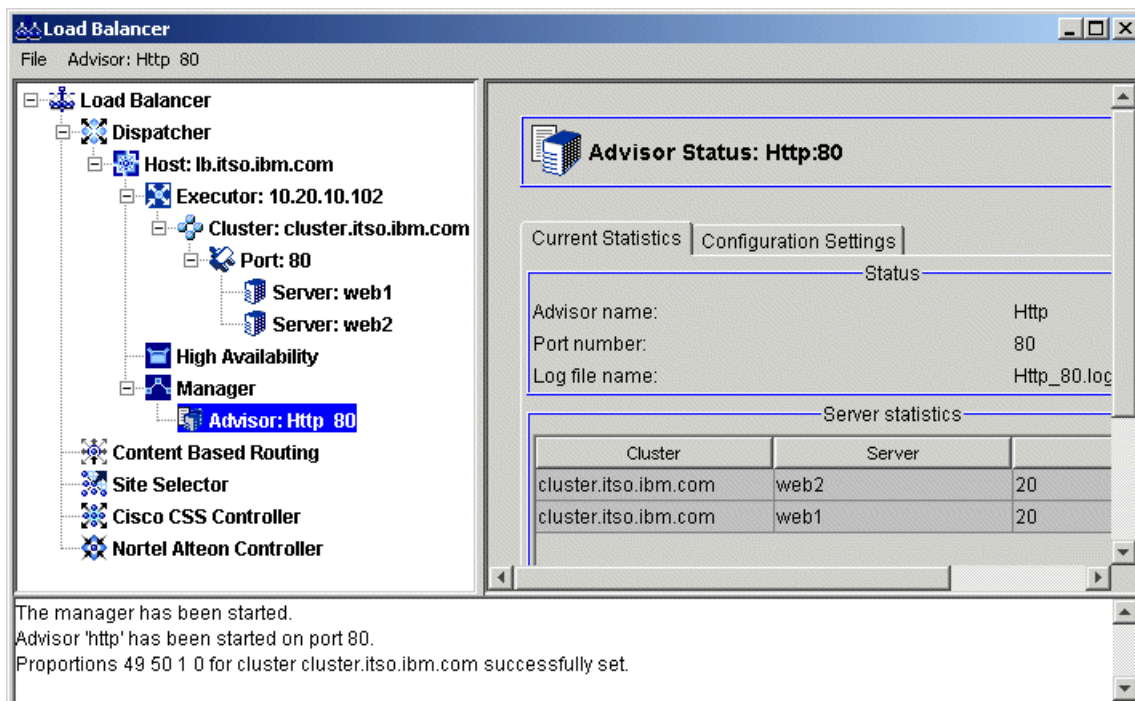


Figure 7-19 Manager started

### 7.4.8 Checking what you have done until now

It might be a good idea to check if everything is working you configured so far. To do so, open a browser on the lb machine and go to the URL:

```
http://cluster.itso.ibm.com/
```

Next, open a browser on the cproxy machine. Configure the browser to use **cproxy.itso.ibm.com** on **port 80** as your connection proxy and try to connect to this again:

```
http://cluster.itso.ibm.com/
```

In both cases you should see the main page of one of the HTTP servers.

## 7.5 Configuring WebSphere clusters

This section explains how to configure WebSphere clusters. In our sample environment, the application is separated into two clusters: one that houses the Web containers, and one that houses the EJB containers. The Web container cluster contains all the servlets and JSP content used to provide presentation services to the application. The EJB cluster houses all the business logic and Enterprise beans that access data services when needed. In addition, we want to provide session failover and thus need to configure persistent session management.

### 7.5.1 Introduction

Before you can configure the clusters and the session management, you have to make sure that the WebSphere Application Server V5.1 software is installed on both application server machines (app1 and app2). In addition, the IBM WebSphere Application Server Network Deployment V5.1 software must be installed on the Deployment Manager machine (dm).

After installing the software, create app1 and app2 and add these application servers to the cell managed by dm. Also, remove the default application server server1 from both nodes. These steps are not explained in this section.

**Note:** For details on how to install and configure WebSphere Application Server V5.1 and IBM WebSphere Application Server Network Deployment V5.1, refer to the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195-00. Chapter 7 explains the Windows installation, while Chapter 8 details the AIX installation steps. See Chapter 13 (Server configuration and management) for application server configuration. Chapters 7 and 8 also explain how to add nodes to a cell for Windows and AIX respectively.

The configuration of the cell should look similar to the configuration shown in Figure 7-20 on page 278. To verify this, log on to the WebSphere Administrative Console and select **System Administration -> Cell**. Select the **Local Topology** tab.

Please note that so far there are no applications deployed on the application servers except for the adminconsole (for using the WebSphere Administrative Console) and filetransfer (for synchronization of the configuration across the nodes) applications. The jmsserver and nodeagent servers are present on both app1 and app2.

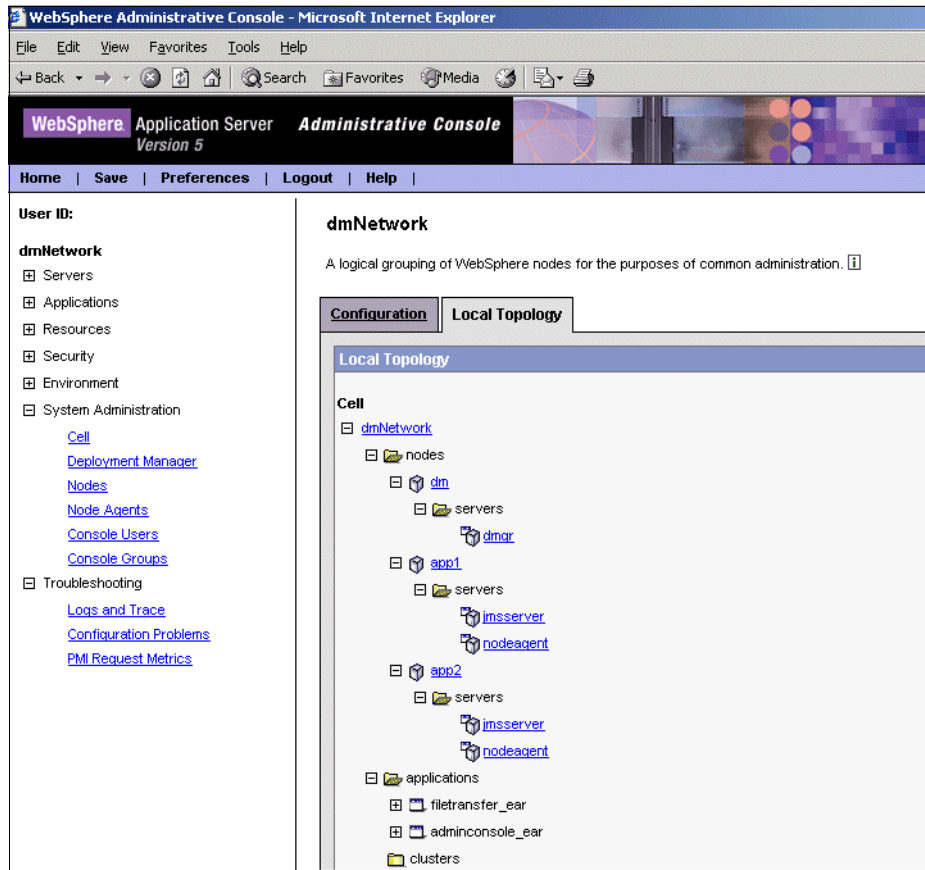


Figure 7-20 The dmNetwork cell base configuration (no clusters configured)

After configuring and/or verifying the correct base configuration, you are now ready to start configuring the ITSO sample topology as detailed in Figure 7-1 on page 258.

A high-level description of the procedure for creating a server cluster and its cluster members follows:

1. Create the original instance of the application server that you want to add to the cluster. Configure it exactly as you would like it. This will be the template for all other application servers. For example, you can deploy enterprise beans into the application server's container and configure the application server to meet specific performance goals.
2. Create a cluster by using the WebSphere Administrative Console Web application.

3. One application server must be added to the cluster. When creating a new application server, server templates that use the definition of existing application servers can be selected, or a previously undefined application server can be created. So you can either choose an existing application server from the list (for example the application server you created in step 1 on page 278) or create a new application server as the first member of the cluster.

**Note:** Server templates are a new feature since WebSphere Application Server V5.0 and are used to duplicate an existing application server definition without adding the application components from the original application.

4. Install the enterprise application into the Deployment Manager.
5. Specify module mappings to place the Web and EJB modules into the appropriate application servers.

## 7.5.2 Creating the Web container cluster

First we describe how to create the Web cluster, which we decided to call “WEBcluster”. As mentioned before, this cluster will be used to provide workload balancing and failover for servlets and JSPs.

Creating a cluster consists of three steps:

1. Step 1 allows you to enter basic cluster information.
2. Step 2 is to define the cluster members. These can be either existing application servers or you can create new ones.
3. Step 3 basically summarizes the information you entered before.

To create a new cluster:

1. Log on to the WebSphere Administrative Console and select **Servers -> Clusters**. In the right pane, a list of clusters defined within the cell is shown. This list should currently be empty.
2. Click **New** to create your first cluster (see Figure 7-21 on page 280). This launches the window for Step 1 of the Create New Cluster process as shown in Figure 7-22 on page 280.

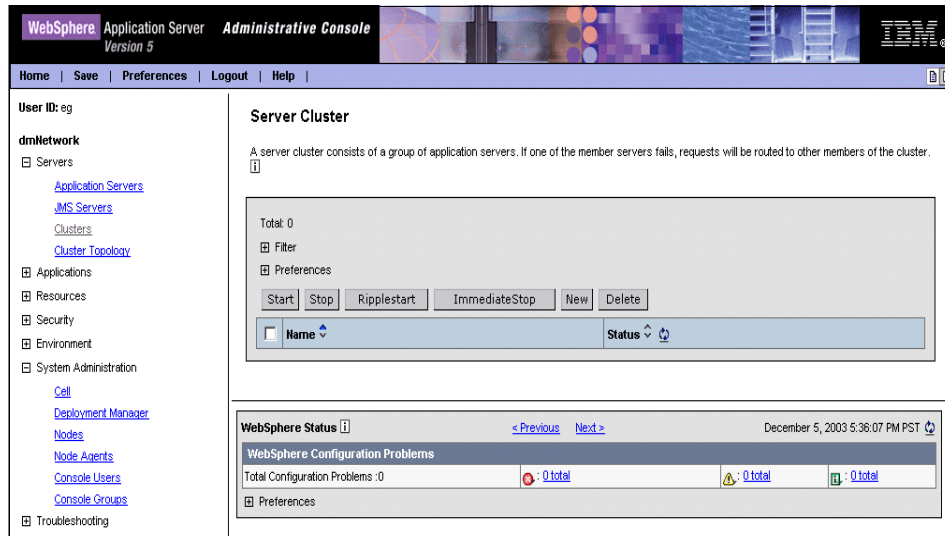


Figure 7-21 Create a new cluster

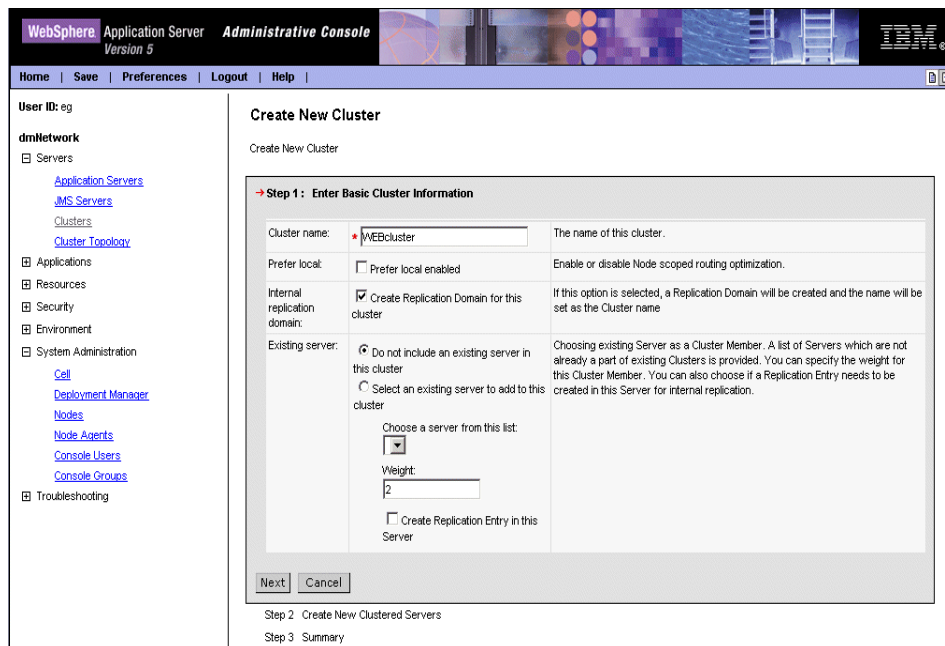


Figure 7-22 Creating a new cluster, Step 1: Enter Basic Cluster Information

3. Now enter the basic cluster information:



- a. Enter **WEBcluster** for the mandatory Cluster name field.
- b. Uncheck the box **Prefer local enabled**. Selecting Prefer local indicates that the EJB running on the local node should be routed the request first if available.

**Important:** We chose to disable the Prefer Local option in order to demonstrate the workload management features of IBM WebSphere Application Server Network Deployment V5.1. However, we recommend this optimization to be enabled in a performance-tuned environment.

- c. Check the box **Create Replication domain for this cluster**.
- d. Accept the default for the Existing server option, and click **Next** to continue. This brings up the window shown in Figure 7-23, which allows you to create new application servers to be added to the cluster.

The screenshot displays the 'Create New Cluster' wizard in the IBM WebSphere Administrative Console. The left-hand navigation pane shows the 'dmNetwork' tree with 'Clusters' selected. The main window is titled 'Create New Cluster' and shows 'Step 2: Create New Clustered Servers'. The form includes the following fields and options:

- Name:** WEB1a (The name of the new cluster member)
- Select Node:** app1 (The new cluster member will be created on the selected node)
- Weight:** 2 (Controls the amount of work directed to the application server. If the weight value for the server is greater than the weight values assigned to other servers in the cluster, then the server will receive a larger share of the servers' workload.)
- Http Ports:** ☒ Generate Unique Http Ports (Generates unique port numbers for every http:transport that is defined in the source server, so that the resulting server that is created will not have HTTP Transports which conflict with the original server or any other servers defined on the same node.)
- Replication entry:** ☒ Create Replication Entry in this Server (If selected, a replication entry will be created for the new cluster member)
- Select template:** ☒ Default application server template (server1) (Cluster members must be created using either a default template or an existing application server as a model. Each member is required to use the same model.)
- ☐ Existing application server

An 'Apply' button is highlighted with a mouse cursor. Below the form, there is a table with the following structure:

Application servers	Nodes	Weight

At the bottom, there are 'Previous', 'Next', and 'Cancel' buttons. The wizard is currently on 'Step 3: Summary'.

Figure 7-23 Creating a new cluster, Step 2: Create New Clustered Servers

4. To create new clustered application servers:
  - a. Enter **WEB1a** for the name of the first new clustered server (cluster member).

- b. Select **app1** from the Select Node drop-down box.
  - c. Check the box **Create Replication Entry in this Server**.
  - d. Accept the default for all other options and click **Apply** to continue.
  - e. Using the same procedure, add a second application server called **WEB1b** to node app1.
  - f. Again, using the same procedure, add a third application server called **WEB2**, but add this server to node app2.
  - g. After adding the three clustered servers, click **Next** to continue.
5. Check the summary (see Figure 7-24) and click **Finish** to create the new cluster.

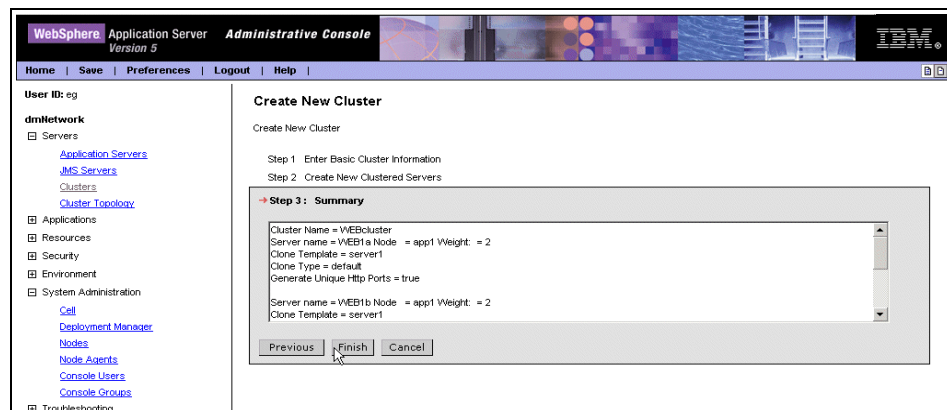


Figure 7-24 Creating a new cluster, Step 3: Summary

6. After completing the above steps, the WebSphere Administrative Console warns that changes have been made to the local configuration. Click **Save** in order to save the changes to the master configuration. Select the **Synchronize changes with Nodes** check box to make sure the changes are synchronized to all nodes in the cell. Click **Save** once again.

**Note:** We created a replication domain for the cluster as well as replication entries for each cluster member. This is done in order to later use WebSphere internal messaging for persistent session support.

WebSphere internal messaging uses a small and fast publish/subscribe engine supplied with WebSphere. It consists of two key elements, a replication domain and replication entries. These are defined at the cell level. A *replication domain* defines the set of replicator processes that communicate with each other and share a common publish/subscribe cluster. A *replicator* runs inside an application server process. The replicator entry defines its basic settings, such as a communication port among replicators and a communication port with clients.

### 7.5.3 Creating the EJB cluster

We continue by creating the EJB cluster, which we shall refer to as “EJBcluster”. This cluster will be used to serve Enterprise beans that access data services when needed, as well as house all the business logic.

1. Log on to the WebSphere Administrative Console and select **Servers -> Clusters**. In the right pane, a list of all clusters defined for the cell is shown. The Web cluster should be listed here. Naturally, the second cluster is also created by clicking **New** (see Figure 7-21 on page 280).
2. Enter the basic cluster information on the Step 1 window of the Create New Cluster process:
  - a. Enter **EJBcluster** for the Cluster name.
  - b. Again, uncheck the box **Prefer local enabled**.
  - c. Accept the default for all other options and click **Next** to launch the Step 2 window of the Create New Cluster process.
3. On the second window of the Create New Cluster process:
  - a. Enter **EJB1a** for the name of the first new clustered server (cluster member).
  - b. Select **app1** from the Select Node drop-down box.
  - c. Accept the default for all other options and click **Apply** to continue.
  - d. Using the same procedure, add another new clustered server called **EJB1b** again on node app1.
  - e. Again using the same procedure, add a third clustered server called **EJB2** but this time add the server to node app2.

4. After adding these three clustered servers, the WebSphere Administrative Console should look as illustrated in Figure 7-25. Click **Next** to continue, then check the summary and click **Finish** to create the EJB cluster.

The screenshot shows the 'Create New Cluster' page in the WebSphere Administrative Console. The page is titled 'Create New Cluster' and is at 'Step 2: Create New Clustered Servers'. It contains a form for adding new cluster members. The form has fields for 'Name', 'Select Node' (a dropdown menu showing 'app2'), 'Weight' (a text box with '2'), 'Http Ports' (a checkbox for 'Generate Unique Http Ports' which is checked), and 'Replication entry' (a checkbox for 'Create Replication Entry in this Server' which is unchecked). Below the form is a table listing existing cluster members. The table has columns for 'Application servers', 'Nodes', and 'Weight'. The table contains three rows: 'EJB1a' with node 'app1' and weight '2', 'EJB1b' with node 'app1' and weight '2', and 'EJB2' with node 'app2' and weight '2'. At the bottom of the page are 'Previous', 'Next', and 'Cancel' buttons. The 'Next' button is highlighted with a mouse cursor.

WebSphere Application Server Administrative Console  
Version 5

Home | Save | Preferences | Logout | Help

### Create New Cluster

Create New Cluster

Step 1 Enter Basic Cluster Information

→ Step 2: Create New Clustered Servers

Enter information about the new server below, and then use the Apply button to add it to the list of cluster members that will be created for this cluster. Use the Edit button to edit the properties of a server already included in the list. Use the Delete button to remove a server from the list.

Name:  The name of the new cluster member

Select Node:  The new cluster member will be created on the selected node

Weight:  Controls the amount of work directed to the application server. If the weight value for the server is greater than the weight values assigned to other servers in the cluster, then the server will receive a larger share of the servers' workload.

Http Ports: ☒ Generate Unique Http Ports Generates unique port numbers for every http transport that is defined in the source server, so that the resulting server that is created will not have HTTP Transports which conflict with the original server or any other servers defined on the same node.

Replication entry: ☐ Create Replication Entry in this Server If selected, a replication entry will be created for the new cluster member

Apply

Edit Delete

Application servers	Nodes	Weight
<input type="checkbox"/> EJB1a	app1	2
<input type="checkbox"/> EJB1b	app1	2
<input type="checkbox"/> EJB2	app2	2

Previous Next Cancel

Step 3 Summary

Figure 7-25 Create EJB cluster

5. After completing these steps, the console warns that changes have been made to the local configuration. Click **Save** in order to save the changes to the master configuration (including synchronization with the cell's nodes).

Now it is time to verify the cluster topology. Using the WebSphere Administrative Console, click **System Administration -> Cell**. Select the **Local Topology** tab. After expanding all items in the right pane, the cluster topology looks as shown in Figure 7-26 on page 285.

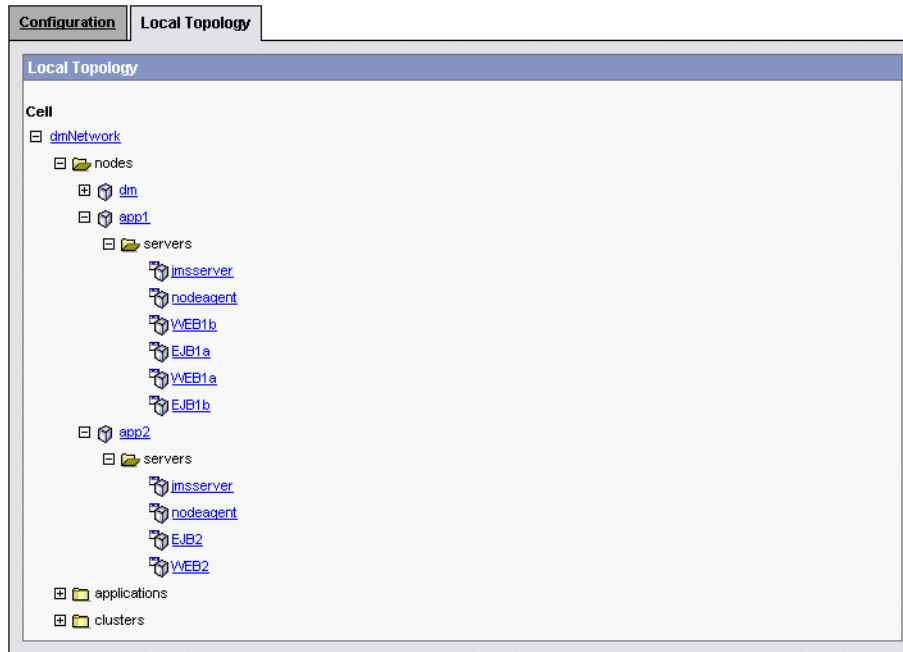


Figure 7-26 The cluster topology

## 7.5.4 Configure persistent session management

WebSphere Application Server V5.1 supports persistent session management in two ways:

- ▶ Database session persistence, where sessions are stored in a defined database. This mechanism was already implemented in WebSphere Application Server V4.0.
- ▶ Memory-to-memory (or in-memory) session replication, where sessions are stored in one or more application server instances.

For more information about distributed session management, see “Session persistence considerations” on page 59. Or refer to Chapter 14 (Configuring session management), especially Section 14.9, “Persistent session management” of the redbook *IBM WebSphere Version 5.0 System Management and Configuration* SG24-6195-00.

For our sample topology, we decided to use memory-to-memory replication. The following section will show you how to configure this function.

When creating our WEBcluster, we also created a replication domain. See 7.5.2, “Creating the Web container cluster” on page 279 for details. Thus, the first thing to do is to verify that the replication domain has been set up:

1. Log on to the WebSphere Administrative Console and select **Environment -> Internal Replication Domains**. You should see the replication domain WEBcluster here (associated with the cluster WEBcluster).
2. Click the **WEBcluster** replication domain. Scroll down and select **Replicator Entries** (listed under Additional Properties). There should be a replicator entry for each application server member within the WEBcluster as shown in Figure 7-27.

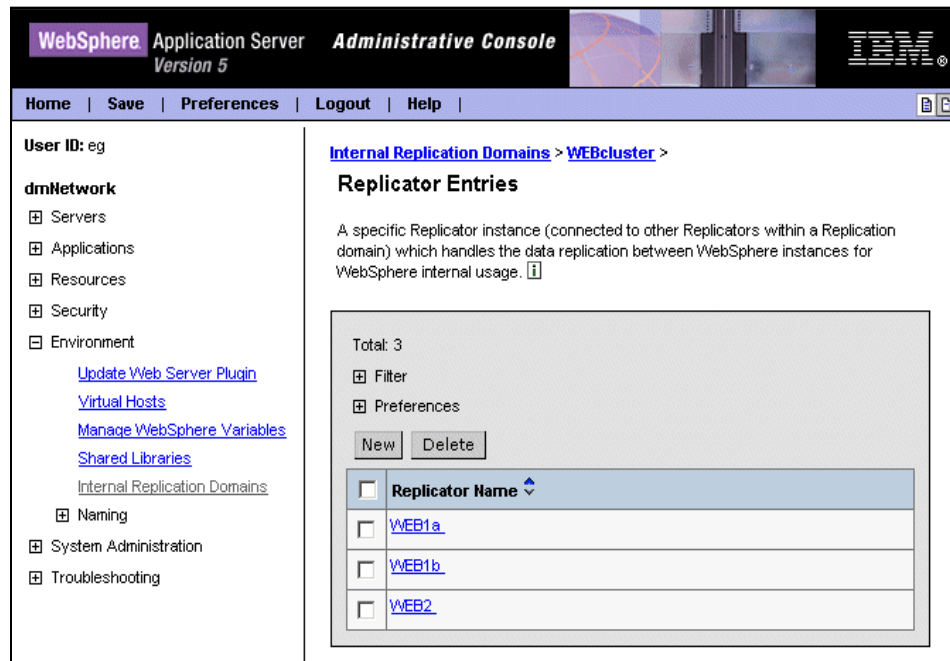


Figure 7-27 Replicator entries for Internal Replication Domain WEBcluster

The following instructions need to be repeated for each application server in the WEBcluster (WEB1a, WEB1b, and WEB2):

1. Using the WebSphere Administrative Console, select **Servers -> Application Servers** and select the first application server (WEB1a) from the right pane.
2. Scroll down to Additional Properties and select **Web Container**.
3. On the next window, in the Additional Properties section, click **Session Management**.

- Again, scroll down to the Additional Properties and select **Distributed Environment Settings**. The window shown in Figure 7-28 is displayed.

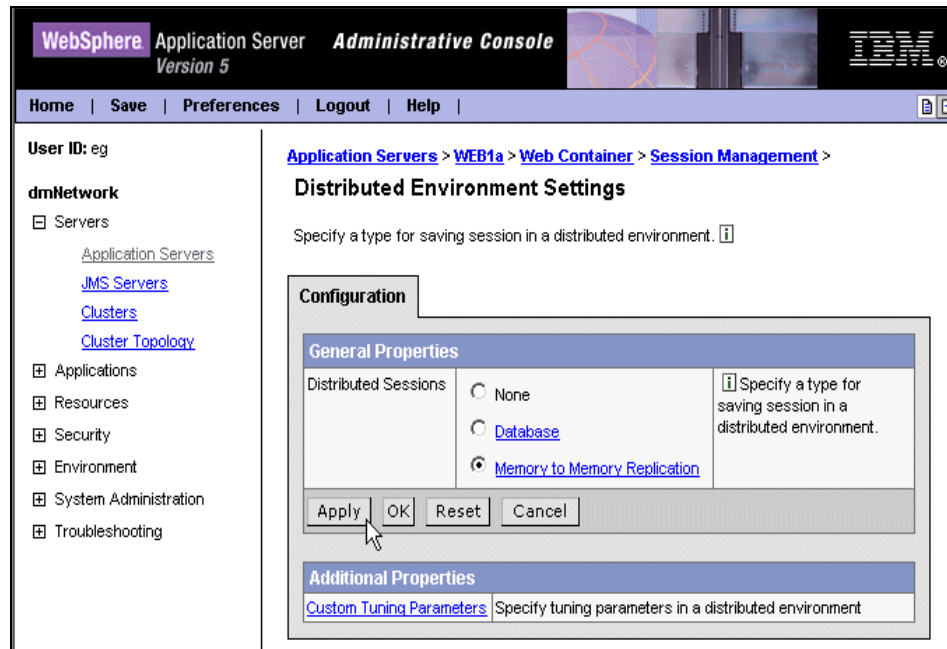


Figure 7-28 Select Memory-to-Memory Replication

- Check the **Memory-to-Memory Replication** radio button and click **Apply**.
- Now click the **Memory-to-Memory Replication** link itself to display the Internal Messaging configuration window shown in Figure 7-29 on page 288.
- Select the **WEBcluster** replicator from the drop-down and ensure the Runtime mode is set to **Both client and server**. Click **Apply** to use this configuration.

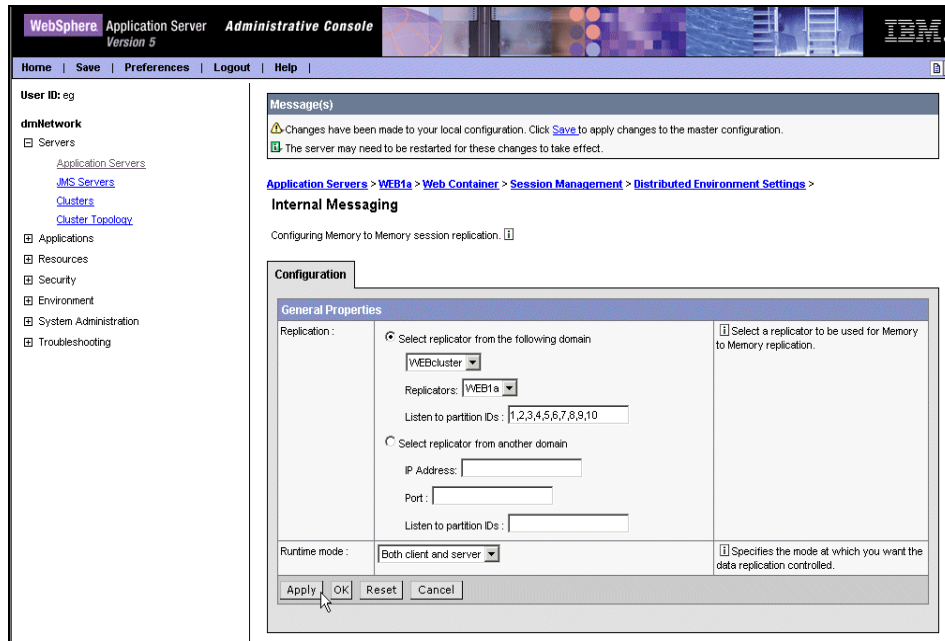


Figure 7-29 Configuring memory-to-memory session replication

8. Click the **Distributed Environment Settings** link on the top right of the window and select **Custom Tuning Parameters** (under Additional Properties).
9. Select **Low (optimize for failover)** for the Tuning level and click **Apply** as detailed in Figure 7-30 on page 289. This will ensure an update of all session data of a particular session before the servlet engine finishes its request.



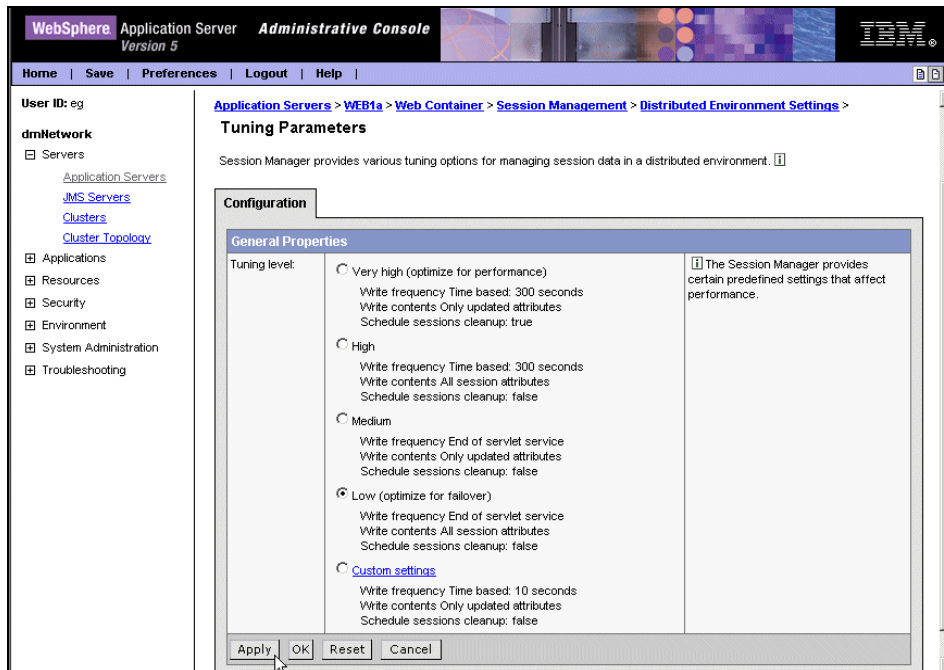


Figure 7-30 Tune the persistent Session Manager

10. Click **Save** to save the changes to the master configuration.
11. Perform step 1 on page 286 to step 10 for each application server within the cluster.

**Important:** In a cluster, by default, sessions are replicated to all servers in the cluster that are connected to the same replicator domain. This replication can be redundant if there is a large number of servers in a cluster. The Session Management facility has an option to partition the servers into groups when storing sessions. There is also an option to replicate the session to only one other server (the Single Replica option in the Internal Replication Domains configuration). When this option is chosen, one single server in the replication domain is selected during session creation, and all updates to the session are only replicated to that single server.

The option we have chosen - each application server holds all session data - should be regarded as the most robust one. However, performance could be affected by this. Thus, in a production environment, you should consider other options as well.

12. Start the newly created clusters via **Servers -> Clusters**. Check both clusters and click the **Start** button. Use the Refresh icon to verify the status.

## 7.6 Installing and configuring BeenThere

The BeenThere application can be used to actually see workload management (WLM) of HTTP requests and of Enterprise JavaBean (EJB) requests. BeenThere consists of the BeenThere servlet, which is used to demonstrate to which application server in a cluster an HTTP request was dispatched and of the BeenThere stateless session EJB used to demonstrate to which application server in a cluster an EJB request was dispatched.

An article by the developer of BeenThere with many details about the code and how it can be used is found at:

<http://www.sys-con.com/websphere/article.cfm?id=321>

Select the **Source Code for this Article** link to see the source code. Scroll down to the end of this page and you find a link called **Additional Code for this Article** that allows you to download the BeenThere code.

In addition, we have placed the BeenThere code into the additional materials repository of this redbook. Refer to Appendix C, “Additional material” on page 935 for instructions on how to obtain it.

After downloading the BeenThere package, extract the .ear file from it. This Enterprise Application Resource is all we need.

### 7.6.1 BeenThere installation summary

The following actions have to be performed in order to set up and install BeenThere:

- ▶ Download the BeenThere package
- ▶ Install the BeenThere application
- ▶ Regenerate the Web server plug-in (plugin-cfg.xml)
- ▶ Restart the servers

### 7.6.2 Install BeenThere

To install and deploy the BeenThere Enterprise Application Archive file, follow these steps:

1. Log on to the WebSphere Administrative Console and click **Applications -> Install New Application**.

2. The Preparing for the application installation window appears as shown in Figure 7-31 on page 291.
  - a. Here you can specify the EAR/WAR/JAR module to upload and install. Click the **Browse** button to specify the location of the BeenThere50.ear file, select it and click **Next** to continue.

**Note:** The .ear file can reside on any node in the cell or on the local workstation running the WebSphere Administrative Console. Depending on where you stored it, use the **Browse** button next to the Local path or Server path selection.

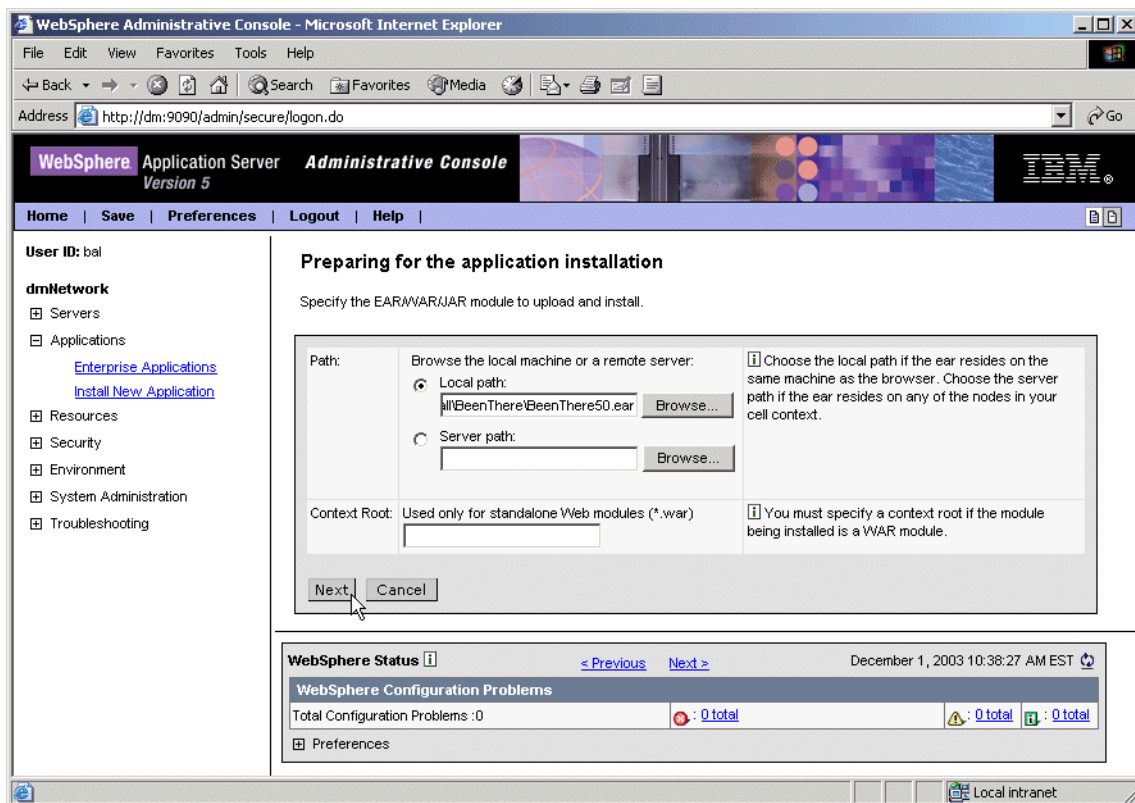


Figure 7-31 Specify the location of the BeenThere ear file

- b. On the next window you can define mappings and bindings. We do not need to change anything here, so click **Next** to accept all defaults.
3. The upcoming window shows the first of several steps in the Install New Application process.

- a. Select to **Deploy EJBs** during the installation.
- b. Correct the Application Name to BeenThere as shown in Figure 7-32 on page 292.

→ Step 1: Provide options to perform the installation

Specify the various options available to prepare and install your application.

AppDeployment Options	Enable
Pre-compile JSP	<input type="checkbox"/>
Directory to Install Application	<input type="text"/>
Distribute Application	<input checked="" type="checkbox"/>
Use Binary Configuration	<input type="checkbox"/>
Deploy EJBs	<input checked="" type="checkbox"/>
Application Name	<input type="text" value="BeenThere"/>
Create MBeans for Resources	<input checked="" type="checkbox"/>
Enable Class Reloading	<input type="checkbox"/>
Reload Interval in Seconds	<input type="text"/>
Deploy WebServices	<input type="checkbox"/>

Figure 7-32 Install New Application process - Deploy EJBs option

- c. Click the link to **Map EJB References to beans**. Here you have to change the JNDI name of BeenThere for the BeenThere WAR module to its fully qualified JNDI name: **cell/clusters/EJBcluster/BeenThere**. This is shown in Figure 7-33.

→ Step 4: Map EJB references to beans

Each EJB reference defined in your application must be mapped to an Enterprise bean.

Module	EJB	URI	Reference Binding	Class	JNDI Name
BeenThere WAR		BeenThere.war\WEB-INF\web.xml	BeenThereBean	com.ibm.websphere.beenthere.BeenThere	<input type="text" value="cell/clusters/EJBcluster/BeenTh"/>

Figure 7-33 Install New Application process - Map EJB references to beans

- d. Click he link to **Map modules to application servers**.
  - i. From the Clusters and Servers selection box, choose **WebSphere:cell=dmNetwork,cluster=WEBcluster**. Then check the box for the module **BeenThere WAR** and click **Apply**.

- ii. Next, select **WebSphere:cell=dmNetwork,cluster=EJBcluster**.  
Check the box for the module **BeenThere EJB** and click **Apply**.

After doing so, the mappings should look as illustrated in Figure 7-34 on page 293.

→ Step 6 : Map modules to application servers

Specify the application server where you want to install modules contained in your application. Modules can be installed on the same server or dispersed among several servers.

Clusters and Servers:

WebSphere:cell=dmNetwork,cluster=WEBcluster  
WebSphere:cell=dmNetwork,cluster=EJBcluster

Apply

<input type="checkbox"/>	Module	URI	Server
<input type="checkbox"/>	BeenThere EJB	BeenThere.jar,META-INF/ejb-jar.xml	WebSphere:cell=dmNetwork,cluster=EJBcluster
<input type="checkbox"/>	BeenThere WAR	BeenThere.war,META-INF/web.xml	WebSphere:cell=dmNetwork,cluster=WEBcluster

Previous

Next

Cancel

Figure 7-34 Install New Application process - Mapping modules to clusters

- e. Click the **Summary** link, review the options and click **Finish** to perform the installation (see Figure 7-35).

→ **Step 8: Summary**

Summary of Install Options

Options	Values
Deploy EJBs Option - RMIC	
Deploy EJBs Option - Classpath	
Distribute Application	Yes
Use Binary Configuration	No
Cell/Node/Server	<a href="#">Click here</a>
Enable Class Reloading	No
Create MBeans for Resources	Yes
Deploy EJBs	Yes
Reload Interval in Seconds	
Application Name:	BeenThere
Directory to Install Application	
Deploy EJBs Option - Database Type	DB2UDB_V81
Pre-compile JSP	No
Application Name	BeenThere
Deploy EJBs Option - Database Schema	
Deploy WebServices	No

Previous Finish Cancel

Figure 7-35 Install New Application process - summary before actual installation

4. **Save** your changes to the master configuration and don't forget to synchronize the changes with the other nodes.

### 7.6.3 Regenerate Web server plug-in

After installing BeenThere, you need to regenerate the Web server plug-in. To do so, log on to the WebSphere Administrative Console and select **Environment -> Update Web Server Plug-in**. Click the **OK** button to update the plug-in configuration file.

After the operation completes, the plugin-cfg.xml will be updated according to the latest configuration. This file is stored in  
 <WAS\_HOME>\config\cells\plugin-cfg.xml on the Deployment Manager machine.

Before distributing the new plug-in file to the two Web servers, you have to change some path information in it:

- Open plugin-cfg.xml in an editor and change the path to the files plugin-key.kdb and plugin-key.sth for all servers. In plugin-cfg.xml, the path will be C:\WebSphere\DeploymentManager\etc.

This should be replaced by **C:\WebSphere\AppServer\etc**, which is the default location of the mentioned files on the Web servers. See Example 7-3.

- After making these changes, copy the modified plugin-cfg.xml to the Web servers. They should be copied to the directory <WAS\_HOME>\config\cells.

---

*Example 7-3 Modifying the generated plugin-cfg.xml*

---

*generated plugin-cfg.xml*

```
<Property name="keyring"
value="C:\WebSphere\DeploymentManager\etc\plugin-key.kdb"/>
<Property name="stashfile"
value="C:\WebSphere\DeploymentManager\etc\plugin-key.sth"/>
```

*modified plugin-cfg.xml*

```
<Property name="keyring" value="C:\WebSphere\AppServer\etc\plugin-key.kdb"/>
<Property name="stashfile"
value="C:\WebSphere\AppServer\etc\plugin-key.sth"/>
```

---

**AIX:** On AIX, you also have to change the plugin-cfg.xml file before distributing it to the Web servers. Change the path of the plugin-key.kdb and plugin-key.sth files from /usr/WebSphere/DeploymentManager/etc/ to **/usr/WebSphere/AppServer/etc/**.

After making the changes, copy the modified plugin-cfg.xml to the Web servers. They have to be copied to /usr/WebSphere/AppServer/config/cells.

Alternatively you can regenerate the Web server plug-in from the command line on the Deployment Manager machine directly to where the Web servers are by specifying a couple of additional parameters as shown in Example 7-4. This example assumes you mapped the Web server as drive "P" on your Windows machine.

---

*Example 7-4 Regenerating Web server plug-in from the Deployment Manager machine*

---

```
C:\WebSphere\DeploymentManager\bin>GenPluginCfg.bat -destination.root
C:\WebSphere\AppServer -destination.operating.system windows -output.file.name
"p:\WebSphere\AppServer\config\cells\plugin-cfg.xml"
```

---

**Note:** On a Unix system, you can regenerate the Web server plug-in as shown in Example 7-5.

*Example 7-5 Generating Web server plug-in for Unix systems*

```
dm:root # GenPluginCfg.bat -destination.root /usr/WebSphere/AppServer  
-destination.operating.system unix -output.file.name  
"/<dest_web_server_config_cell>/plugin-cfg.xml"
```

## 7.6.4 Restart servers

After the BeenThere installation has completed, you have to restart the following servers:

► Web servers

Log on as Administrator and click **Start -> Settings -> Control Panel -> Administrative Tools -> Services**. In the Services window, select **IBM HTTP Server 2.0.42.2**. Right-click and select **Restart**.

**AIX:** On AIX, do the following to restart a Web server: Log on as root user and run the following command:

```
/usr/IBMHttpServer/bin/apachectl restart
```

► WEBcluster

While still logged on to the WebSphere Administrative Console, click **Servers -> Clusters** and select the cluster called WEBcluster. Click **Stop** to stop it. Once it has stopped, select the cluster again and click **Start**.

► EJBcluster

Repeat these steps to restart the EJBcluster.

## 7.6.5 Verifying BeenThere

The last step is to verify that BeenThere is working correctly. Follow these steps to verify the configuration:

1. Open the BeenThere start page. Point your browser to the following URL:

`http://<your_caching_proxy>/wlm/beenthere`

In our environment:

`http://cproxy.itso.ibm.com/wlm/beenthere`



**Notes:**

- ▶ cproxy.itso.ibm.com resolves to the IP address of our caching proxy (10.20.10.101). The caching proxy then points to the Web server cluster (10.20.10.100) associated with the Load Balancer machine (10.20.10.101).
- ▶ If you encounter problems accessing BeenThere through your Web server or caching proxy, you can try to access BeenThere directly via your application server's embedded HTTP transport using:

`http://<your_app_server>:9080/wlm/beenthere`

In our environment, the URL is:

`http://app1.itso.ibm.com:9080/wlm/beenthere`

The port number depends on your configuration.

2. Enter **6** for the number of Bean Iterations and click **Execute**.

The result should look like Figure 7-36 on page 298. As you can see, the request has first gone to server app2, then the six EJB requests have been workload managed across the EJB cluster members on both application server nodes app1 and app2. This demonstrated EJB weighted round robin workload management.

Remember that there was only one request to the Web cluster, so you cannot see Web container workload management on the BeenThere page.

3. Reload the page without changing the number of Bean Iterations. You should see that the next request to the Web cluster has been workload managed. The servlet node should now be app1 instead of app2.

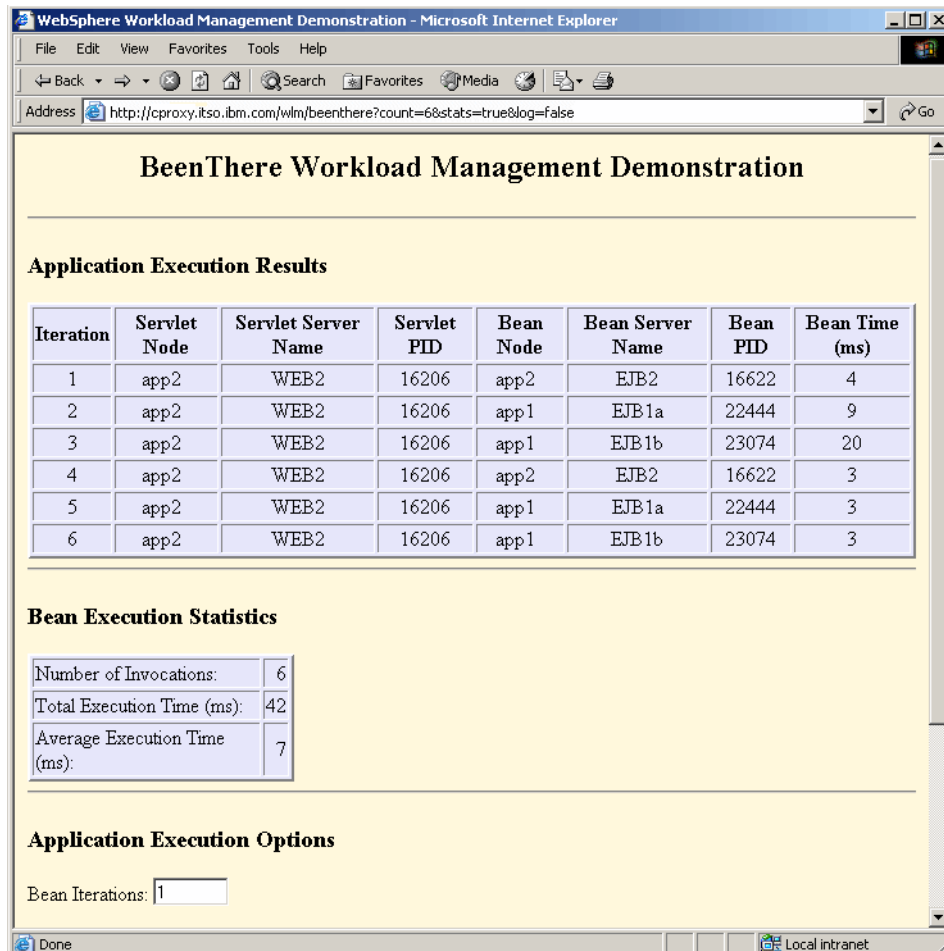


Figure 7-36 BeenThere demonstrating EJB workload management

## 7.7 Installing and configuring Trade3.1

We will now describe how to install and configure Trade3.1 into the ITSO sample topology (shown in Figure 7-1 on page 258). Compared to the lightweight BeenThere application, Trade3.1 is much more complex.

### 7.7.1 Trade3.1 installation summary

The following actions have to be performed in order to set up and install Trade3.1:

- ▶ Download the Trade3.1 package
- ▶ Set up and configure Trade3DB database
- ▶ Create JDBC and JMS resources
- ▶ SOAPify Trade3.ear
- ▶ Install the application
- ▶ Regenerate the Web server plug-in (plugin-cfg.xml)
- ▶ Restart servers

### 7.7.2 Download the Trade3.1 package

You can download the Trade3.1 code from the following Web site:

<http://www.ibm.com/software/webservers/appserv/was/performance.html>

After downloading the file trade3Install.zip, unzip the package on the Deployment Manager machine. A directory called t3install will be created. This directory contains all Trade3.1 files.

Trade3.1 uses a database to store its data. Thus you have to create a DB2 database called “Trade3db” on the database server and DB2 client access must be set up on each application server node.

**Attention:** At the time of writing this book, Trade3.1 was not yet available for download from the Web site mentioned above. Only Trade3 (for WebSphere Application Server V5.0) was downloadable. However, the updated code will become available in the near future.

Please note that you *cannot* use Trade3 with a WebSphere Application Server V5.1 installation, this version can *only* be used for WebSphere Application Server V5.0.

### 7.7.3 Set up and configure Trade3DB database

#### DB2 Server

We will use the default DB2 instance on the server (db2inst1).

1. In a Windows environment, log on to the database server and start a DB2 shell using the **db2cmd** command.

**AIX:** On AIX, first switch to user db2inst1:

```
su - db2inst1
```

2. Copy the file table.ddl from the Trade3.1 install directory to the DB2 server.  
Then execute the following DB2 commands to create the Trade3db database:

```
db2 create db Trade3db
db2 connect to Trade3db
db2 -tvf table.ddl
db2 disconnect all
db2set DB2_RR_TO_RS=yes
db2 update db config for Trade3db using logfilsiz 1000
db2 update db cfg for Trade3db using maxappls 100
db2stop force
db2start
```

3. From the same shell, change the current directory to *<sqllib>\bnd*, in our case to C:\Program Files\Sql\lib\bnd. Run the following commands to bind DB2 packages for the Trade3db database:

```
db2 connect to Trade3db
db2 bind @db2cli.lst blocking all grant public
```

**AIX:** On AIX, change to the following directory before performing the above commands:

```
/usr/opt/db2_08_01/bnd
```

**Important:** Failure to rebind will result in the following runtime error:

```
COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver][DB2/NT] SQL0805N
Package "NULLID.SQLLC300" was not found. SQLSTATE=51002
```

## DB2 clients

The next step is to configure the database clients installed *on both application server nodes* (app1 and app2 respectively).

**Tip:** For some reason, if your commands don't work, please try them with the double quotation mark, for example:

```
db2 "connect to Trade3db user <db_user> using <password>"
```

1. Log on to the application server machine(s) as the DB2 instance owner and start a db2 shell.

2. Using this shell, you can now catalog the remote database server db using the command:

```
db2 catalog tcpip node <your_local_node> remote <db2_server_hostname>  
server <port_from_the_services_file_on_your_OS_(50000)>
```

So in our case, the correct command is:

```
db2 catalog tcpip node db remote db server 50000
```

**Note:** Please note that the host name (db) must be resolvable on the DB2 client machine.

3. Next you have to catalog the remote database Trade3db:

```
db2 catalog database Trade3db as Trade3db at node <your_local_node>
```

So in our case, the correct command is:

```
db2 catalog database Trade3db as Trade3db at node db
```

4. Verify the database connectivity from the DB2 client machines (app1, app2):

```
db2 connect to Trade3db user <db_user> using <password>
```

In our case, the following command will result in displaying the information shown in Example 7-6:

```
db2 connect to Trade3db user db2inst1 using <password>
```

*Example 7-6 Database connectivity from DB2 clients to DB2 server*

---

Database Connection Information

Database server	= DB2/6000 8.1.3
SQL authorization ID	= DB2INST1
Local database alias	= Trade3DB

---

## 7.7.4 Create JDBC and JMS resources

In order to prepare your J2EE environment to run Trade3.1, you have to create JDBC and JMS resources. For example, on each application server node you have to define a JDBC data source that points to the database Trade3db you just created.

These steps can be performed using the WebSphere Administrative Console. However, Trade3.1 comes with .jacl scripts specifically for clustered environments. We will use these script files.

1. Open up a command prompt window on the Deployment Manager machine.

2. Ensure C:\WebSphere\DeploymentManager\bin is in your PATH. If not, add it using:

```
SET PATH=%PATH%;C:\WebSphere\DeploymentManager\bin
```

3. Run the following command:

```
wsadmin -f Trade3.jacl config_install
```

**AIX:** On AIX, perform the following steps to run the script:

1. Log on as root user
2. Ensure /usr/WebSphere/DeploymentManager/bin is included in your PATH. If not, add it using:

```
export PATH=$PATH:/usr/WebSphere/DeploymentManager/bin
```

3. Now run the following command:

```
/wsadmin.sh -f Trade3.jacl config_install
```

## 7.7.5 SOAPify Trade3.ear

Running Trade3 in the SOAP runtime mode requires running the WebSphere SOAPEarEnabler on the Trade3 Enterprise Application Archive (EAR).

1. Open a Windows command prompt window on the Deployment Manager machine.
2. Ensure C:\WebSphere\DeploymentManager\bin is in your PATH.
3. Change to the directory containing the Trade3.1 files (t3install) and issue the following command:

```
SoapEarEnabler.bat Trade3.ear 1 Soapdd.xml n 1 Trade3EJB.jar n /soap
```

**AIX:** On AIX, perform the following steps to run the script.

1. Log on as root user.
2. Ensure /usr/WebSphere/DeploymentManager/bin is included in your PATH.
3. Change to the directory containing the Trade3.1 files (t3install) and run the following command:

```
SoapEarEnabler.sh Trade3.ear 1 Soapdd.xml n 1 Trade3EJB.jar n /soap
```

## 7.7.6 Install Trade3.1 from the WebSphere Administrative Console

You are now ready to install the Trade3.1 Enterprise Application Archive.

1. Log on to the WebSphere Administrative Console and click **Applications -> Install New Application**. The Preparing for the application installation

window is displayed (compare to the BeenThere installation, Figure 7-31 on page 291).

- a. First you have to specify the EAR/WAR/JAR module to upload and install. Select **Browse** to specify the location of the SOAP-enabled Trade3.ear file, select it, and click **Next** to continue.

**Note:** The .ear file can reside on any node in the cell or on the local workstation running the WebSphere Administrative Console. Depending on where you stored it, click the **Browse** button next to the Local path or Server path selection.

- b. On the next window you can define mappings and bindings. We do not need to change anything here, so click **Next** to accept all defaults.
  - c. Click the **Continue** button on the Application Security Warnings window.
2. The upcoming window shows the first of 12 steps in the Install New Application process.
    - a. On the Step 1 window, make sure the **Deploy EJBs** check box is selected. Please note that the Application Name Trade3 has been filled in automatically.

**Note:** If you wish to install Trade3.1 into a directory other than config\apps, then you can specify the correct path in the Directory to Install Application field.

- b. Click **Step 7** (Map EJB References to beans, see Figure 7-37 on page 304).
  - i. Change the JNDI name of Trade for the TradeWeb module to its fully qualified JNDI name:  
`cell/clusters/<EJB_Cluster_Name>/Trade`  
In our environment, the correct fully qualified JNDI name is:  
`cell/clusters/EJBcluster/Trade`
  - ii. Change the JNDI name of Quote for the TradeWeb module to its fully qualified JNDI name:  
`cell/clusters/<EJB_Cluster_Name>/Quote`  
In our environment, the correct fully qualified JNDI name is:  
`cell/clusters/EJBcluster/Quote`

→ **Step 7 : Map EJB references to beans**

Each EJB reference defined in your application must be mapped to an Enterprise bean.

Module	EJB	URI	Reference Binding	Class	JNDI Name
TradeEJBs	TradeEJB	trade3EJB.jar,META-INF/ejb-jar.xml	ejb/Trade	com.ibm.websphere.samples.trade.ejb.Trade	<input type="text" value="Trade"/>
TradeEJBs	TradeEJB	trade3EJB.jar,META-INF/ejb-jar.xml	ejb/Quote	com.ibm.websphere.samples.trade.ejb.LocalQuote	<input type="text" value="Quote"/>
TradeEJBs	TradeEJB	trade3EJB.jar,META-INF/ejb-jar.xml	ejb/Account	com.ibm.websphere.samples.trade.ejb.LocalAccount	<input type="text" value="Account"/>
TradeEJBs	TradeEJB	trade3EJB.jar,META-INF/ejb-jar.xml	ejb/Holding	com.ibm.websphere.samples.trade.ejb.LocalHolding	<input type="text" value="Holding"/>
TradeEJBs	TradeEJB	trade3EJB.jar,META-INF/ejb-jar.xml	ejb/Order	com.ibm.websphere.samples.trade.ejb.LocalOrder	<input type="text" value="Order"/>
TradeEJBs	TradeEJB	trade3EJB.jar,META-INF/ejb-jar.xml	ejb/KeySequence	com.ibm.websphere.samples.trade.ejb.LocalKeySequence	<input type="text" value="KeySequence"/>
TradeEJBs	TradeEJB	trade3EJB.jar,META-INF/ejb-jar.xml	ejb/AccountProfile	com.ibm.websphere.samples.trade.ejb.LocalAccountProfile	<input type="text" value="AccountProfile"/>
TradeEJBs	TradeBrokerMDB	trade3EJB.jar,META-INF/ejb-jar.xml	ejb/Trade	com.ibm.websphere.samples.trade.ejb.Trade	<input type="text" value="Trade"/>
TradeEJBs	KeySequenceEJB	trade3EJB.jar,META-INF/ejb-jar.xml	ejb/KeyGen	com.ibm.websphere.samples.trade.ejb.LocalKeyGen	<input type="text" value="KeyGen"/>
TradeEJBs	AccountEJB	trade3EJB.jar,META-INF/ejb-jar.xml	ejb/AccountProfile	com.ibm.websphere.samples.trade.ejb.LocalAccountProfile	<input type="text" value="AccountProfile"/>
TradeWeb		trade3Web.war,WEB-INF/web.xml	ejb/Trade	com.ibm.websphere.samples.trade.ejb.Trade	<input type="text" value="cell/clusters/EJBcluster/Trade"/>
TradeWeb		trade3Web.war,WEB-INF/web.xml	ejb/Quote	com.ibm.websphere.samples.trade.ejb.Quote	<input type="text" value="cell/clusters/EJBcluster/Quote"/>
TradeWeb		trade3Web.war,WEB-INF/web.xml	ejb/LocalQuote	com.ibm.websphere.samples.trade.ejb.LocalQuote	<input type="text" value="Quote"/>
TradeWeb		trade3Web.war,WEB-INF/web.xml	ejb/LocalAccountHome	com.ibm.websphere.samples.trade.ejb.LocalAccount	<input type="text" value="Account"/>

Figure 7-37 Provide fully qualified JNDI names for the TradeWeb module level

- c. Click **Step 10** (Map virtual hosts for Web modules). Ensure that both TradeWeb and Apache-SOAP are mapped onto the Virtual Host default\_host.
- d. Click **Step 11** (Map modules to application servers, see Figure 7-38 on page 305) and map the EJB and Web application modules to their respective cluster.
  - i. From Clusters and Servers, select **WebSphere:cell=dmNetwork,cluster=WEBcluster**. Check the box for the module TradeWeb and click **Apply**.
  - ii. Next, select **WebSphere:cell=dmNetwork,cluster=EJBcluster**. Check the box for the module **TradeEJBs** and click **Apply**.

Your mappings should now look as illustrated in Figure 7-38 on page 305.



**→ Step 11: Map modules to application servers**

Specify the application server where you want to install modules contained in your application. Modules can be installed on the same server or dispersed among several servers.

WebSphere: cell=dmNetwork,cluster=EJBcluster  
WebSphere: cell=dmNetwork,cluster=WEBcluster

Clusters and Servers:

<input type="checkbox"/>	Module	URI	Server
<input type="checkbox"/>	TradeEJBs	trade3EJB.jar,META-INF/ejb-jar.xml	WebSphere: cell=dm1wNetwork,cluster=EJBcluster
<input type="checkbox"/>	TradeWeb	trade3Web.war,WEB-INF/web.xml	WebSphere: cell=dm1wNetwork,cluster=WEBcluster
<input type="checkbox"/>	Apache-SOAP	soap.war,WEB-INF/web.xml	WebSphere: cell=dm1wNetwork,cluster=WEBcluster

Figure 7-38 Mapping modules to separate clusters

- e. Click **Step 12 (Summary)**, review the options and click **Finish** to perform the installation (see Figure 7-39). This can take a few minutes.

**WebSphere Application Server Administrative Console**  
Version 5

Home | Save | Preferences | Logout | Help

User ID: admin2

**dmNetwork**

- ☒ Servers
- ☐ Applications
  - [Enterprise Applications](#)
  - [Install New Application](#)
- ☐ Resources
- ☐ Security
- ☐ Environment
- ☐ System Administration
- ☐ Troubleshooting

**→ Step 12: Summary**

Summary of Install Options

Options	Values
Deploy EJBs Option - RMIC	
Deploy EJBs Option - Classpath	
Distribute Application	No
Use Binary Configuration	No
Cell/Node/Server	<a href="#">Click here</a>
Enable Class Reloading	No
Create MBeans for Resources	No
Deploy EJBs	Yes
Reload Interval in Seconds	
Application Name:	Trade3
Directory to Install Application	
Deploy EJBs Option - Database Type	DB2UDB_V81
Pre-compile JSP	No
Application Name	Trade3
Deploy EJBs Option - Database Schema	
Deploy WebServices	No

Figure 7-39 Summary before installing the Trade3.1 Enterprise Application

3. Finally, after the installation has completed, save your changes to the master configuration.

## 7.7.7 Regenerate Web server plug-in

After installing Trade3.1 you need to regenerate the Web server plug-in. The procedure is the same as shown in “Regenerate Web server plug-in” on page 294.

## 7.7.8 Restart servers

After the installation is complete, you have to restart the following servers:

- ▶ Web servers

Log on as Administrator and click **Start -> Settings -> Control Panel -> Administrative Tools -> Services**. In the Services window, select **IBM HTTP Server 4.0.42.2**. Right-click and select **Restart**.

**Note:** On AIX, to restart a Web server log on as root user and run the following command:

```
/usr/IBMHttpServer/bin/apachectl restart
```

- ▶ JMS Server on app1

Log on to the WebSphere Administrative Console and click **Servers -> JMS Servers**. Next, select the server **jmsserver** on node app1 and click **Stop**. When it has stopped, select it again and click **Start**.

- ▶ WEBcluster

While still logged on to the WebSphere Administrative Console, select **Servers -> Clusters** and select the cluster WEBcluster. Click **Stop**. When it has stopped, select it again and click **Start**.

- ▶ EJBcluster

Next, go to **Servers -> Clusters** and stop and restart the EJBcluster.

## 7.7.9 Install Trade3.1 using the installation script

You can alternatively use the installation script that comes with Trade3 to install Trade3 into a cluster. The cluster is created while running the installation script. Perform the following steps:

1. Open a Windows command window and change to the **C:\software\t3install** directory (assuming the files were extracted to this directory).

**AIX:** Change to (assuming the files were extracted to this directory):

```
/tmp/t3install
```

2. Run the WebSphere Application Server Network Deployment setupCmdLine script to setup your shell environment.

```
C:\software\t3install>C:\WebSphere\DeploymentManager\bin\setupCmdLine
```

**AIX:** On AIX, do the following:

```
/tmp/t3install # /usr/WebSphere/DeploymentManager/bin/setupCmdLine.bat
```

3. Start the Deployment Manager:

```
C:\software\t3install>C:\WebSphere\DeploymentManager\bin\startManager
```

**AIX:** On AIX, do the following:

```
/tmp/t3install # /usr/WebSphere/DeploymentManager/bin/startManager
```

4. Start the Node Agent:

```
C:\software\t3install>C:\WebSphere\AppServer\bin\startNode
```

**AIX:** On AIX, do the following:

```
/tmp/t3install # /usr/WebSphere/AppServer/bin/startNode
```

5. Install Trade3 JDBC/JMS resources and application interactively:

```
C:\software\t3install>C:\WebSphere\DeploymentManager\bin\wsadmin -f  
Trade3.jacl config_install
```

**AIX:** On AIX, do the following:

```
/tmp/t3install # /usr/WebSphere/DeploymentManager/bin/wsadmin -f  
Trade3.jacl config_install
```

A dialog launches where you need to enter the following settings:

- a. Enter a new cluster name: **TradeCluster**
- b. Enter your node name. This means the node that contains a JMS server.
- c. Enter your node name again. This means the node that contains a cluster member.
- d. Press the **Enter** key.
- e. Enter the DataBase type: **db2**
- f. Enter the JDBC Driver Class Path: **C:\SQLLIB\java\db2java.zip**

**Note:** On AIX, the JDBC Driver Class Path may look like:

`/home/db2inst1/sqllib/java/db2java.zip`

- g. Enter your DB2 userid: `<your_db2_user>` (**db2admin**)
- h. Enter your DB2 password: `<password>` (**db2admin**)
- i. Enter **y** which means install the Trade3 application.

The needed resources are created and the Trade3 application is deployed. The TradeCluster, TradeClusterServer1, JDBC resources, and JMS resources should be created.

- 6. Start the **jmsserver** through the Administrative Console (Servers -> JMS Servers) and verify there are no errors in the jmsserver log.
- 7. Start the TradeCluster (Servers -> Clusters), and verify there no errors in the server logs.
- 8. Verify that Trade3 is started (Applications -> Enterprise Applications).

**Note:** During the installation using the Trade3 configuration script we created only one cluster member. If you want more cluster members, you can add them through the Administrative Console (Servers -> Clusters -> `<cluster_name>` -> Cluster members -> New).

## 7.7.10 Working with Trade3.1

The last step is to verify the Trade3.1 installation. First you should test if Trade3 runs on your application server without using the entire topology, that is using the embedded HTTP transport of your application server.

Open the Trade3.1 start page using the following URL (the port number depends on your configuration):

`http://<host_name>:9080/trade`

In our environment, the URL is:

`http://app1.itso.ibm.com:9080/trade`

If this works, then in a second step you should test using the whole topology, that is, through the Caching Proxy, Load Balancer, and Web server cluster using:

`http://<your_caching_proxy>/trade`

In our environment, the URL is:

`http://cproxy.itso.ibm.com/trade/`

**Note:** cproxy.itso.ibm.com resolves to the IP address of our caching proxy (10.20.10.101). The caching proxy then points to the Web server cluster (10.20.10.100) associated with the Load Balancer machine (10.20.10.101).

This will display the Trade3.1 index page as shown in Figure 7-40.

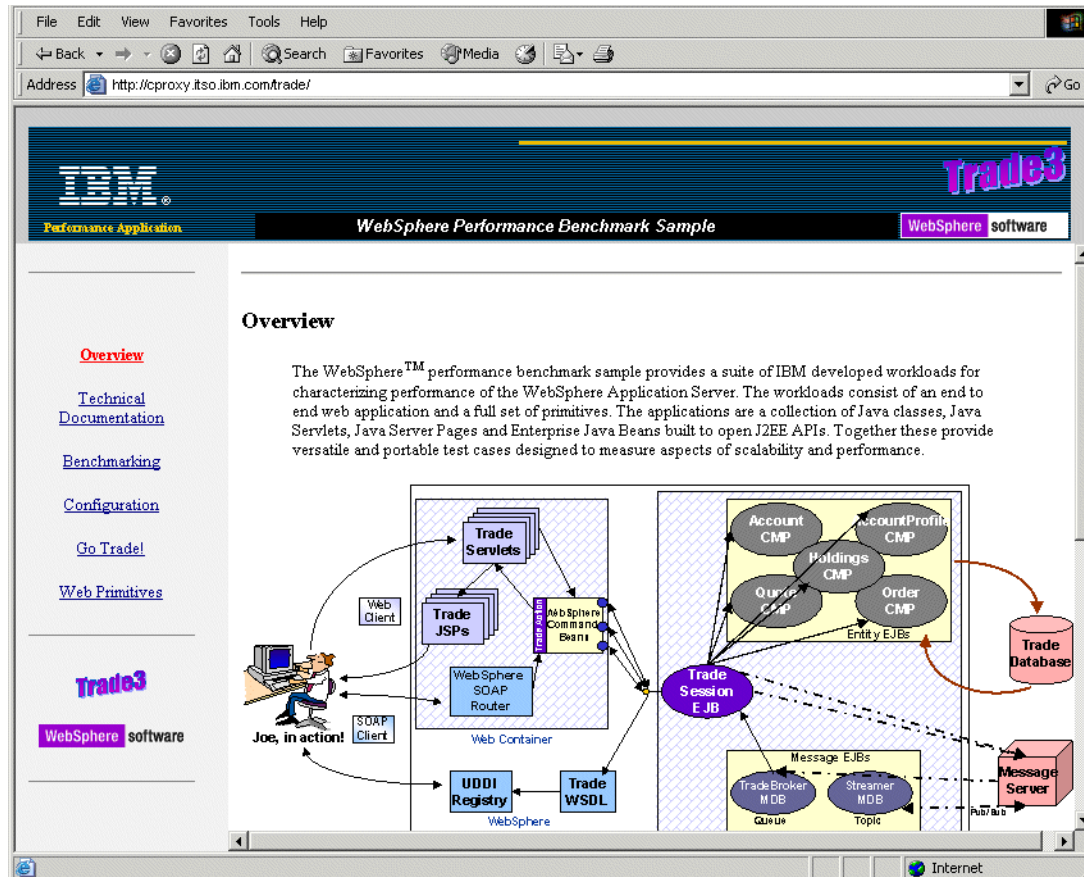


Figure 7-40 Trade3.1 index page

Before you can start using the Trade3.1 application, you first have to populate the database:

1. Click the **Configuration** link on the left-hand side of the Trade3.1 index page.
2. In the Configuration utilities window, click **(Re-)populate Trade Database**. Wait until the database population has been completed.

3. Before closing the window, review the default Trade3.1 configuration and click the **Update Config** button.
4. After populating the database, run the following DB2 commands to update DB2 statistics:

```
db2 connect to Trade3db user <db_user> using <password>
db2 reorgchk update statistics
```

This will improve DB2 performance.

5. Click the **Go Trade!** link on the left-hand side of the Trade3.1 index page. You will be forwarded to the Trade3.1 login page shown in Figure 7-41.
6. Click the **Log in** button and start to use the Trade3.1 application.

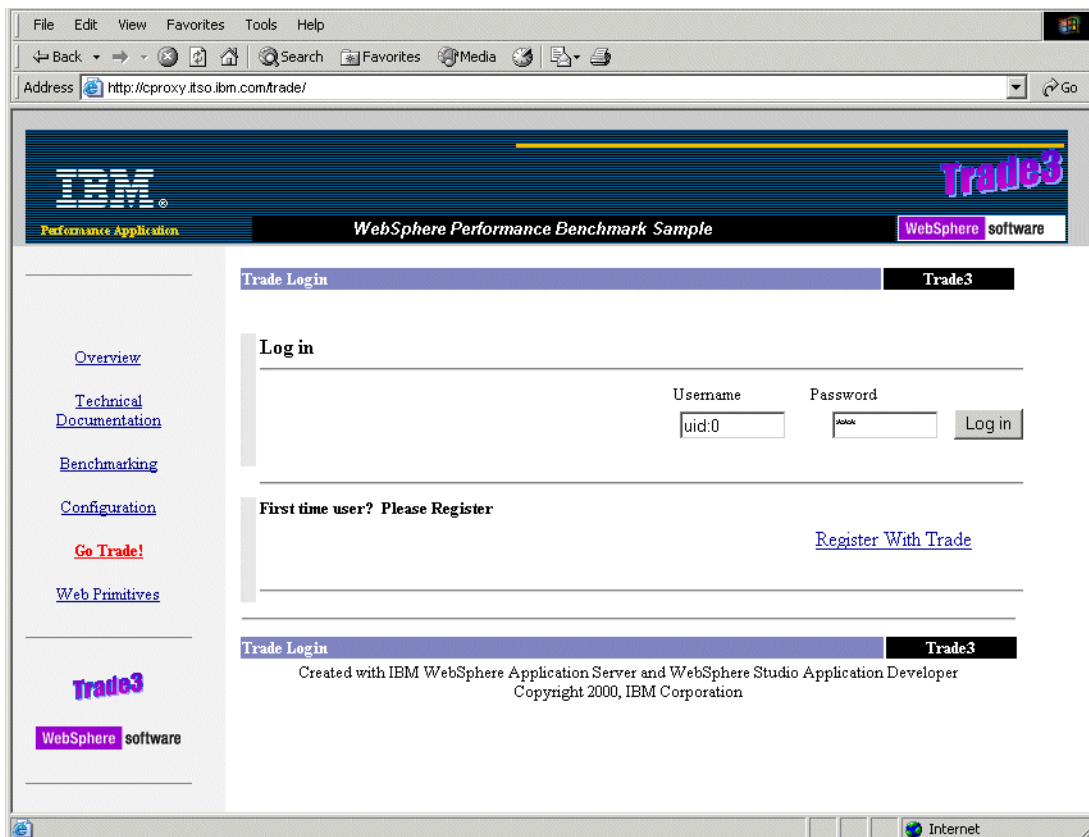


Figure 7-41 Trade3.1 Login page

### 7.7.11 Verify failover with Trade3.1

Now that you have Trade3.1 up and running, verify WebSphere's failover functionality. The following outlines a scenario you could use to do so.

Before you begin, ensure that all servers are started.

1. Log into Trade3.1.
2. On one of the Webcluster servers (WEB1a, WEB1b, or WEB2) a session has been created. Stop two of the Web servers, for example WEB1a and WEB2:  
Open up another browser window, log on to the WebSphere Administrative Console and select **Servers -> Application Servers**. Select the two application servers WEB1a and WEB2 and click **Stop**.
3. Continue using Trade3.1; this will force WEB1b to handle all your requests to the Trade3.1 application.

You are now certain that your session lives in the WEB1b server.

To verify WebSphere failover, you can now restart another Web server of the WEBcluster, for example WEB2. After this server has been restarted, you can then stop WEB1b:

4. Select application server WEB2 and click **Start**. Wait until it has started.
5. Now **stop** application server WEB1b.
6. After it has stopped, continue using Trade3.1, your session will still exist and you can continue using Trade3.1 without any interruptions.

This proves that session persistence is working on the WEBcluster. Another scenario could be to stop one or more EJBcluster application servers, or you could try shutting down an entire node.

### 7.7.12 Volume testing Trade3.1

In order to test the throughput and scalability of a particular application or hardware architecture, you will need to simulate a load. There are a number of tools available for this purpose. Some are available for free and some are not. Refer to 19.2, "Tools of the trade" on page 823 for instructions on how to do this.







## Part 4

# High availability solutions





## High availability concepts

This chapter discusses the concepts of WebSphere system high availability, the different levels of WebSphere end-to-end high availability, and WebSphere high availability solutions planning and implementation.

Maintaining high levels of access to information across heterogeneous environments, without compromising a quality user experience, can challenge any IT organization. In previous chapters, we have discussed WebSphere scalability in some detail. In this chapter and following chapters we discuss WebSphere system high availability.

The content of this chapter has not changed since the previous version of this redbook.

## 8.1 Process availability and data availability

There are two kinds of high availability: process high availability and data high availability.

For example, in a WebSphere system, we have the following processes:

- ▶ WebSphere Application Server process
- ▶ WebSphere Node Agent process
- ▶ WebSphere Deployment Manager process
- ▶ WebSphere JMS process
- ▶ WebSphere MQ process
- ▶ Various database server processes
- ▶ HTTP server process
- ▶ Load Balancer process
- ▶ LDAP server process
- ▶ Firewall process
- ▶ Networked file system (for example NFS, AFS®) process
- ▶ Operation system processes

Even if we make all of these processes highly available, the WebSphere system may still fail due to data availability. Without data availability, the WebSphere system cannot do any meaningful work. Therefore we will discuss how to make the WebSphere data management systems highly scalable and highly available.

Data management is an integral part of a WebSphere production system. It is used for storing:

- ▶ Administrative repository for the WebSphere administrative servers
- ▶ Application data for the WebSphere Application Servers
- ▶ Persistent data for Entity EJBs
- ▶ Persistent session data for WebSphere HTTP sessions
- ▶ WebSphere security data for the LDAP servers
- ▶ Transactional log files and other log files
- ▶ HTML and image files
- ▶ WebSphere system binaries
- ▶ Application binaries
- ▶ Other important data

## 8.2 Clustering for high availability

Clustering is a fundamental approach for accomplishing high availability. IBM WebSphere Application Server Network Deployment V5.1 has a built-in server clustering technique (WLM). In addition, there are many other clustering techniques that can be used for WebSphere end-to-end system high availability.

In order to achieve 99.x% of WebSphere system availability, we need to integrate platform-specific clustering solutions with WebSphere to meet the high availability needs for critical applications. Data access is a very important part of most applications, especially transactional applications. Protecting data integrity and enhancing availability of the entire WebSphere processing environment is achieved by integrating WebSphere and platform-specific clustering software.

We should consider WebSphere high availability as end-to-end system availability; this system includes a database, LDAP, firewall, Load Balancer, HTTP server, and WebSphere server. This system is integrated with platform-specific clustering software to achieve maximum availability.

Generally speaking, the percentage of the failed client requests over the total client requests is very small, and most client request failures are caused by database failures, since the database failover takes minutes to finish if IP failover is used. WebSphere process failures usually do not contribute much to the total client request failures, as WebSphere process failover is instantaneous under the mechanism of the WebSphere workload management. Some parallel database servers such as Oracle Parallel Server (OPS) or Real Application Cluster (RAC) and some techniques such as Transparent Application Failover (TAF) can be used to minimize the total client failures.

There are several platform-clustering software packages available. The unit of failover usually includes a collection of network definitions and disk storage, and one or more services such as DB2 database server, Oracle database server, HTTP server, Firewall, LDAP server, WebSphere Application Server, WebSphere Node Agent or Deployment Manager. However, it is not standardized, and different vendors use different terms. For example, the unit of failover in Sun Cluster is called a *logical host*, while the unit of failover in IBM HACMP or HACMP/ES Cluster is called an *application server*, and the unit of failover in HP MC/ServiceGuard is called a *package*.

There are two kinds of cluster failovers:

- ▶ IP-based cluster failover:

For example, HACMP, HACMP/ES, MC/ServiceGuard, Sun Cluster, VERITAS Cluster, Microsoft Cluster, DB2 Parallel Server (IBM DB2 UDB Enterprise Extended Edition)

- ▶ Non-IP cluster failover:

For example, WebSphere WLM, Oracle Parallel Server, Oracle Real Application Cluster

Usually, IP-based cluster failover is slow (one to five minutes), and non-IP cluster failover is very fast (instantaneous). However, non-IP cluster failover, such as Oracle Parallel Server, still relies on cluster software such as MC/ServiceGuard, Sun Cluster, HACMP, or MSCS to provide the cluster information. Oracle integrated the clustering manager into its products in Oracle 9i on Windows.

In this part, we cover clustering considerations, the failover process, clustering configurations, sample scripts, each WebSphere component's reactions to the failures, and how to tune the cluster system to enhance the performance. We discuss:

- ▶ HACMP and HACMP/ES
- ▶ MC/ServiceGuard
- ▶ Sun Cluster
- ▶ VERITAS Cluster
- ▶ Microsoft Cluster
- ▶ Oracle Parallel Server and Real Application Cluster, both Connection-Time Failover and Transparent Application Failover
- ▶ DB2 Parallel Server Cluster with hardware-based clustering
- ▶ WebSphere Application Server high availability
- ▶ WebSphere Node Agent and Deployment Manager high availability
- ▶ WebSphere high availability enhancements
- ▶ WebSphere JMS server high availability
- ▶ WebSphere MQ high availability
- ▶ Web server high availability
- ▶ Load Balancer high availability
- ▶ Firewall high availability
- ▶ LDAP high availability
- ▶ Highly available network file system

We also address the techniques that are needed to build an end-to-end, highly available WebSphere production system.

There are two types of highly available data services:

- ▶ A failover data service runs an application on only one primary node in the cluster at a time. Other nodes might run other applications, but each application runs on only a single node. If a primary node fails, the applications running on the failed node fail over to another node and continue running. We can further divide this category into two subcategories:
  - Active/Active mode, where two services reside in two nodes that are configured as mutual failover.
  - Active/Standby mode, where one node is configured as the primary to run the service, while the other node is configured as a hot standby.

The configuration for both modes is very similar. The advantage of the Active/Active mode configuration is lower hardware cost. However, the service performance is reduced when a failover occurs. The advantage of the Active/Standby mode configuration is steady performance, but redundant hardware is needed. Furthermore, the Active/Active mode configuration may have twice as many interruptions as the Active/Standby mode configuration, since a failure in any of two nodes may cause a failover.

- A scalable data service spreads an application across multiple nodes to create a single, logical service. Scalable services leverage the number of nodes and processors in the entire cluster on which they run.

## 8.3 Availability definition

Before we describe different high availability (HA) implementations for WebSphere systems, we first need to define high availability and discuss how to measure high availability. Availability is a measure of the time that a server is functioning normally, as well as a measure of the time the recovery process requires after the system fails. In other words, it is the downtime that defines system availability. This downtime includes both planned and unplanned downtime.

Let A be an index of system availability expressed as a percentage, MTBF the mean time between failures, and MTTR the maximum time to recover the system from failures; then we have:

$$A = \text{MTBF} / (\text{MTBF} + \text{MTTR})$$

As MTBF gets larger, A increases and MTTR has less impact on A. As MTTR approaches zero, A increases toward 100 percent. This means that if we can recover from failures very quickly, we will have a highly available system. The time to recover a system includes fault detection time and system recovery time. Therefore, clustering software uses fault detection mechanisms and automatically fails over the services to a healthy host to minimize the fault detection time and the service recovery time. MTTR is minimized because the fault detection time is minimized and no repair attempt is needed. Therefore, A is significantly raised. Any repairs to the failed node and any upgrades of software and hardware will not impact the service availability. This is the so-called *hot replacement* or rolling upgrade.

The availability issue is not as simple as the formula discussed above. First, MTBF is just a trend. For example, if a CPU has an MTBF of 500,000 hours, it does not mean that this CPU will fail after 57 years of use; this CPU can fail at any time. Second, there are many components in a system, and every

component has a different MTBF and MTTR. These variations make system availability unpredictable using the formula above. We can build a simulation model for an end-to-end WebSphere system's availability with random process theory such as Markov chains, but this is beyond the scope of this book.

For a WebSphere production system, the availability becomes much more complicated, since a WebSphere system includes many components, such as firewall, Load Balancer (LB), Web server, WebSphere Application Server and administrative servers (Node Agent and Deployment Manager), administrative repository, JMS server, log files, session persistent database, application database, and LDAP server and database.

Usually, redundant hardware and clustering software are used to achieve high availability. Our goal is to minimize the MTTR through various clustering techniques; if MTTR=0, A=100% no matter what the MTBF is. Using this approach, system availability becomes predictable and manageable.

### **8.3.1 Levels of availability**

First of all, availability is closely related to cost, as shown in Figure 8-1 on page 321. It is important to balance the downtime with cost. The more you invest, the less downtime there is. Therefore, it is also very important for you to evaluate what you will lose if your WebSphere service is temporarily unavailable. Different businesses have different costs for downtime, and some businesses such as financial services may lose millions of dollars for each hour of downtime during business hours. Costs for the downtime include not only direct dollar losses but also reputation and customer relationships losses.



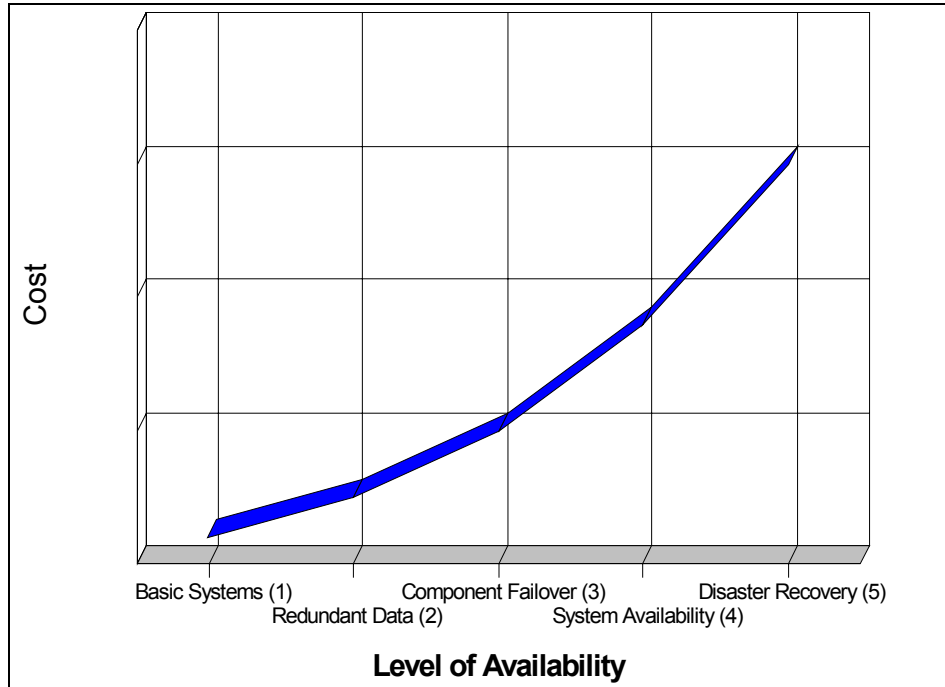


Figure 8-1 Levels of availability and costs

As we discussed above, redundant hardware and clustering software are approaches to high availability. We can divide availability into the following levels:

1. Basic systems. Basic systems do not employ any special measures to protect data and services, although backups are taken regularly. When an outage occurs, the support personnel restores the system from backup (usually tape).
2. Redundant data. Disk redundancy and/or disk mirroring are used to protect the data against the loss of a disk. Full disk mirroring provides more data protection than RAID-5.
3. Component failover. For an e-business infrastructure like WebSphere, there are many components. As we discussed above, an outage in any component may result in service interruption. Multiple threads or multiple instances can be employed for availability purposes. For example, if we do not make the firewall component highly available, it may cause the whole system to go

down (worse than that, it may expose your system to hackers) even though the servers are highly available.

For WebSphere, we have process high availability (vertical scaling) and process and node high availability (horizontal scaling). Entity EJBs are persisted into the database. Highly available data management is critical for a highly available transactional system. Therefore, it is very important to balance the availability of all components in the WebSphere production system. Do not overspend on any particular component, and do not underspend on other components, either. For example, for the system shown in Figure 8-2, the system availability seen by the client would be 85%.

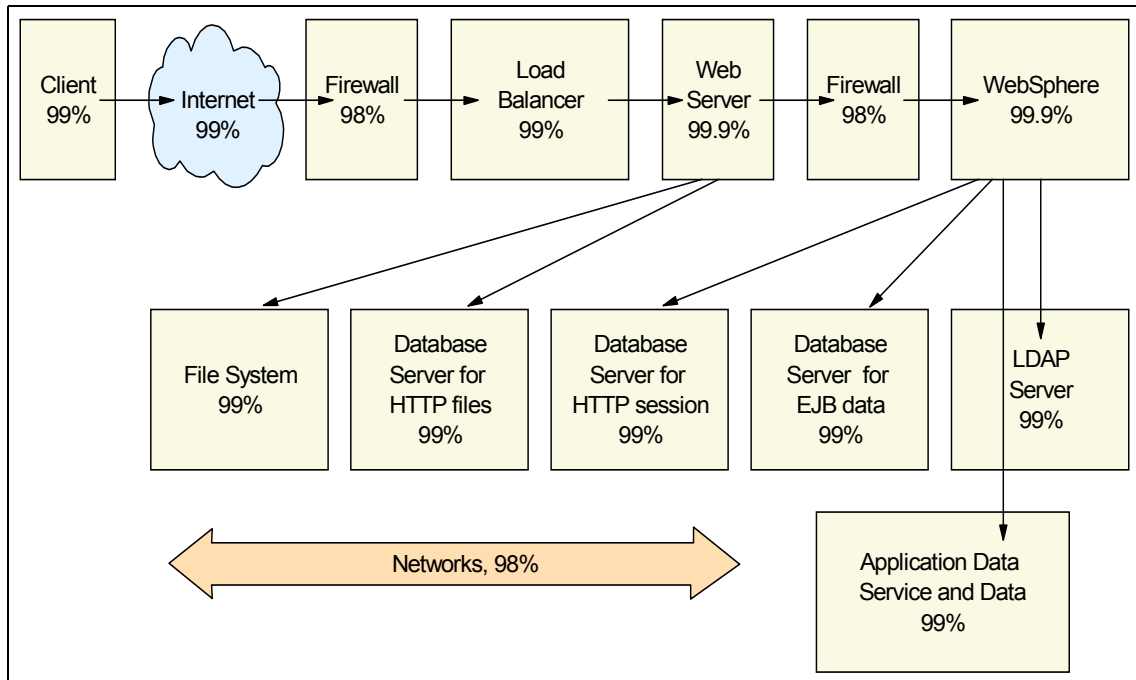


Figure 8-2 Availability chains

4. System failover. A standby or backup system is used to take over for the primary system if the primary system fails. In principle, any kind of service can become highly available by employing system failover techniques. However, this will not work if the software is hard-coded to physical host-dependent variables. We can configure the systems as active/active mutual takeover or active/standby takeover. Although the active/active mutual takeover configuration increases the usage of hardware, it also increases the possibility of interruption, and hence reduces the availability. In addition, it is not efficient to include all components into a single cluster system. We have a firewall cluster, LDAP cluster, WebSphere server cluster, and database

cluster. In system failover, clustering software monitors the health of the network, hardware, and software process, detects and communicates any fault, and automatically fails over the service and associated resources to a healthy host. Therefore, you can continue the service before you repair the failed system. As we discussed before, as MTTR approaches zero, A increases toward 100%. System failover can also be used for planned software and hardware maintenance and upgrades.

5. Disaster recovery. This applies to maintaining systems in different sites. When the primary site becomes unavailable due to disasters, the backup site can become operational within a reasonable time. This can be done manually through regular data backups, or automatically by geographical clustering software.

Continuous availability means that high availability *and* continuous operations are required to eliminate all planned downtime.

Different customers have different availability requirements and downtime costs. For example, one hour of downtime can cost millions of dollars for brokerage firms, but only tens of thousands of dollars for transportation firms. Furthermore, system availability is determined by the weakest point in the WebSphere production environment.

### 8.3.2 Availability matrix

We all talk about the uptime, and everybody wants 100% uptime. In reality, a 100% uptime system is prohibitively expensive to implement, as we discussed previously. For some applications, 99% uptime is adequate, leaving a downtime of 14 minutes per day on average (see Table 8-1 on page 324). For some applications, 99.9% or higher uptime is required. Many people refer to 99%, 99.9%, 99.99%, and 99.999% as *two nines*, *three nines*, *four nines*, and *five nines*. The *five nines* is generally thought of as the best achievable system with reasonable costs, and many vendors offer such solutions. These vendors include:

- ▶ IBM with WebSphere WLM and Cluster
- ▶ IBM with WebSphere MQ Cluster
- ▶ IBM with HACMP on AIX
- ▶ HP with MC/ServiceGuard on HP-UX
- ▶ Sun Microsystems with Sun Cluster on Solaris
- ▶ VERITAS with VERITAS Cluster Server
- ▶ Microsoft with Microsoft Cluster Server on Windows
- ▶ Oracle Parallel Server or Real Application Cluster
- ▶ DB2 Parallel Server

We describe the implementation of a WebSphere HA system with all of these products in the next sections.

*Table 8-1 Availability matrix - “nine” rule*

<b>9s</b>	<b>Percentage of uptime</b>	<b>Downtime per year</b>	<b>Downtime per week</b>	<b>Downtime per day</b>
	90%	36.5 days	16.9 hours	2.4 hours
	95%	18.3 days	8.4 hours	1.2 hours
	98%	7.3 days	3.4 hours	28.8 minutes
Two 9s	99%	3.7 days	1.7 hours	14.4 minutes
	99.5%	1.8 days	50.4 minutes	7.2 minutes
	99.8%	17.5 hours	20.2 minutes	2.9 minutes
Three 9s	99.9%	8.8 hours	10.1 minutes	1.4 minutes
Four 9s	99.99%	52.5 minutes	1 minute	8.6 seconds
Five 9s	99.999%	5.3 minutes	6 seconds	864 milliseconds
Six 9s	99.9999%	31.5 seconds	604.8 milliseconds	86.4 milliseconds
Seven 9s	99.99999%	3.2 seconds	60.5 milliseconds	8.6 milliseconds
Eight 9s	99.999999%	315.4 milliseconds	6 milliseconds	0.9 milliseconds

The five nines availability allows a downtime of 864 milliseconds per day, 6 seconds per week, and 5.3 minutes per year (see Table 8-1). For all of these clustering techniques with IP takeover, a typical database failover takes two to three minutes, so MTTR=2.5 minutes. We therefore need an MTBF of 183 days to achieve 99.999% availability. That means only two failovers per year. For Oracle Parallel Server/Real Application Cluster, failover can be done in seconds because the instance is pre-existing and IP takeover is not needed.

Some businesses require 7x24x365 availability, while others require 6x20 or 5x12 availability. The latter do not reduce the requirement for high availability if the business requires the minimum interruption during its business hours. Since we do not know when outages will happen, clustering techniques can keep MTTR short and increase available time even if a business operates only 5x12.

Even though clustering techniques can keep a service highly available, service performance may degrade after the failure occurs until the failed system rejoins the cluster after repair.

Therefore, we suggest describing availability using three factors:

- ▶ System uptime percentage
- ▶ Business operation hours and pattern
- ▶ Performance availability requirement

We should design a high availability system to satisfy the uptime requirement during operation hours and to meet the performance availability requirement.

Most business applications do not require 7x24, so software and hardware upgrades can be performed in the scheduled maintenance time. For the business that requires 7x24 services, clustering techniques provide rolling upgrades and hot replacement by manually failing over from one system to another.

### 8.3.3 Causes of downtime

The causes of downtime can be either planned events or unplanned events. As shown in Table 8-2, planned events account for as much as 30% of downtime. As we have discussed above, rolling upgrade and hot replacement can reduce the planned downtime. However, the most important issue is how to minimize the unplanned downtime, since nobody knows when the unplanned downtime occurs and all businesses require the system to be up during business hours. From Table 8-2, we can see that software failures are responsible for 40% of system downtime. Software failures include network software failure, server software failure, and client software failure.

Table 8-2 Causes of downtime

Cause of downtime	Percentage
Software failures	40%
Hardware failures	10%
Human errors	15%
Environmental problems	5%
Planned downtime	30%

The human error factor is also a major cause of downtime. Although education is important, it is also important or perhaps even more important to design easy-to-use system management facilities.

The end-to-end WebSphere high availability system that eliminates a single point of failure (SPOF) for all parts of the system can minimize both planned and unplanned downtime. We will describe the implementation of such a WebSphere high availability system.

Many environmental problems are data center related. Having a locally located standby does not suffice, because the entire site environment may be affected. Geographic clustering and data replication can minimize downtime caused by such environmental problems.

### 8.3.4 Possible single points of failure in the WebSphere system

Table 8-3 lists potential single points of failure in the WebSphere system and possible solutions.

*Table 8-3 Possible SPOF in the WebSphere system*

Failure point	Possible solutions	Note
Entity EJBs, Application DB	HA DBs, parallel DBs	Catch <code>StaleConnectionException</code> and retry
Log files	HA DBs, parallel DBs, clustering, HA files system, disk mirroring, RAID-5	
LDAP	HA LDAP, master-replica, clustering, sprayer	Manual recovery
WebSphere Deployment Manager	Multiple WebSphere domains (cells), OS-service, hardware-based clustering	
WebSphere Master Repository Data	HA shared file system, Networked file system, hardware based clustering	
WebSphere Node Agent	Multiple Node Agents with WebSphere built-in HA LSD, OS-service, hardware-based clustering	
WebSphere Application Server	EJB WLM, servlet clustering, hardware-based clustering	Use different application servers/nodes for EJBs and servlets

Failure point	Possible solutions	Note
Web server	Multiple Web servers with network sprayer, hardware-based clustering	
Load balancer	HA network sprayers	
Firewall	Firewall clustering, HA firewall, firewall sprayer	
Internal network	Dual internal networks	
Hubs	Multiple interconnected network paths	
NIC failures	Multiple NICs in a host	
Disk failures	Disk mirroring, RAID-5	
Disk bus failure	Multiple buses	
Disk controller failure	Multiple disk controllers	
Network service failures (DNS, ARP, DHCP, etc.)	Multiple DNS, etc. network services	
OS or other software crashes	Clustering, automatically switch to a healthy node	
Host dies	Clustering, automatically switch to a healthy node	
Power outages	Two-power systems	
Room disaster (fire, flood, etc.)	Put the system in a different room	
Floor disasters (fire, flood, etc.)	Put the system on a different floor	
Building disasters (fire, flood, tornado, etc.)	Put the system in a different building	
City disasters (earthquake, flood, etc.)	Remote mirror, replication, geographical clustering	
Region disasters	Put two data centers far away with geographical clustering or remote mirroring	

Failure point	Possible solutions	Note
People error	Train people, simplify system management, use clustering and redundant hardware/software, disable DDL operations	
Software upgrades	Rolling upgrades with clustering and/or WLM for 7x24x365, planned maintenance for others	For 7x24x365, suggest using two-domain WebSphere Application Server; single domain with WLM works, but is not suggested.
Hardware upgrades	Rolling upgrades with clustering and/or WLM for 7x24x365, planned maintenance for others	

### 8.3.5 Levels of WebSphere system availability

We can deploy WebSphere systems with different redundant hardware, software, networks, processes and components. For the convenience of our discussions, we can roughly divide the deployment into several availability levels.

#### WebSphere system HA level 1

For WebSphere system HA level 1, as shown in Figure 8-3 on page 329, WebSphere, the HTTP server, and the database are installed in a single box host. If a host machine dies or crashes, all WebSphere services will be unavailable. A failure of any component such as a network card, network cable, HTTP server, application server, administrative servers (Node Agent and Deployment Manager), JMS server, or database server will cause the WebSphere service to become unavailable. This level is usually used by developers and lowest-end customers.



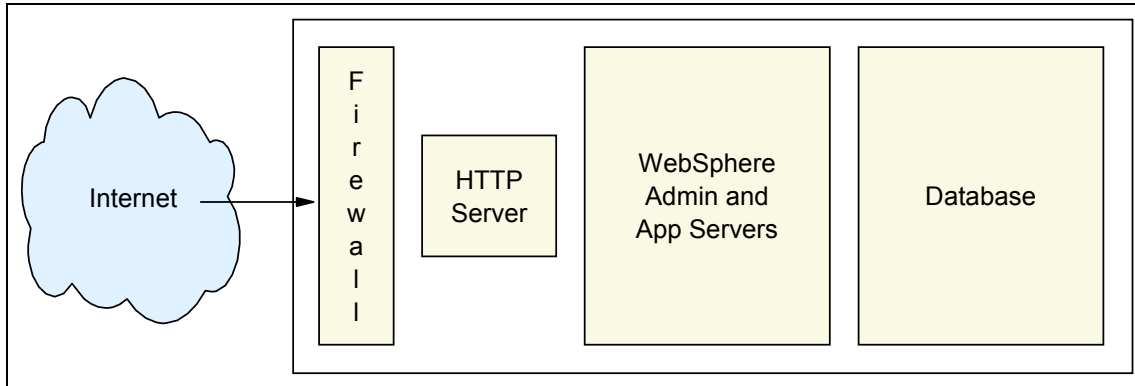


Figure 8-3 WebSphere system HA level 1

With a more powerful machine, you can create a server cluster (with multiple cluster members) that can minimize failures in the application server process.

Single points of failure include the host, firewall, HTTP server, application server (if not clustered), administrative servers (Node Agent and Deployment Manager), JMS server, database server and data, and network.

## WebSphere system HA level 2

For WebSphere system HA level 2, as shown in Figure 8-4 on page 330, the database host is separated from the WebSphere servers. If the database host fails, WebSphere still can service static and cached contents. Again, you can create clusters of WebSphere Application Servers.

Single points of failure include the firewall, HTTP server, application server (if not clustered), administrative servers (Node Agent and Deployment Manager), JMS server, database server and data, and network.

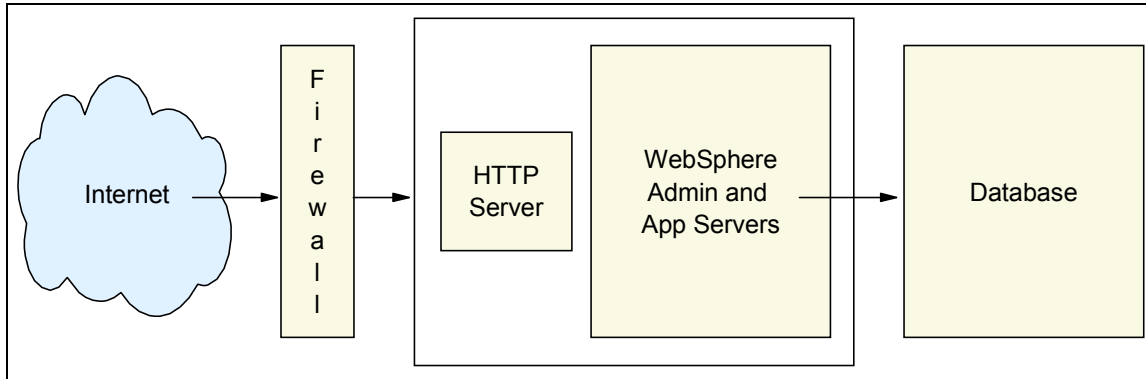


Figure 8-4 WebSphere system HA level 2

### WebSphere system HA level 3

For WebSphere HA level 3, multiple HTTP servers, WebSphere vertical and horizontal scaling, and LDAP are used, as shown in Figure 8-5. Multiple WebSphere administrative domains (cells) can be used to minimize the impacts of WebSphere upgrades.

Single points of failure includes the firewall, sprayer, persistent session data, Entity EJB data, application data, log files, JMS server, Deployment Manager, LDAP and network.

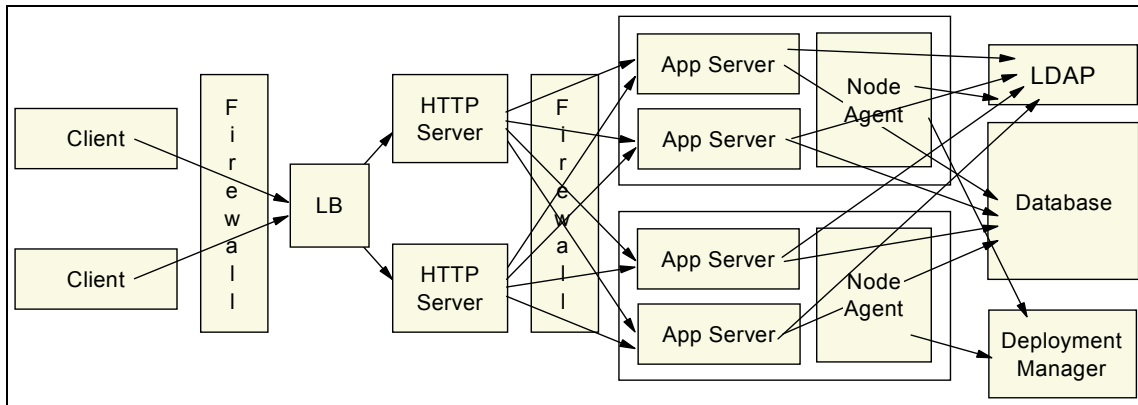


Figure 8-5 WebSphere system HA level 3

### WebSphere system HA level 4

For WebSphere system HA level 4, as shown in Figure 8-6 on page 331, single points of failure are eliminated in each component of the WebSphere system in a site. We will discuss how to implement this level in a later section.

Multiple WebSphere administrative domains can be used for availability during WebSphere upgrades.

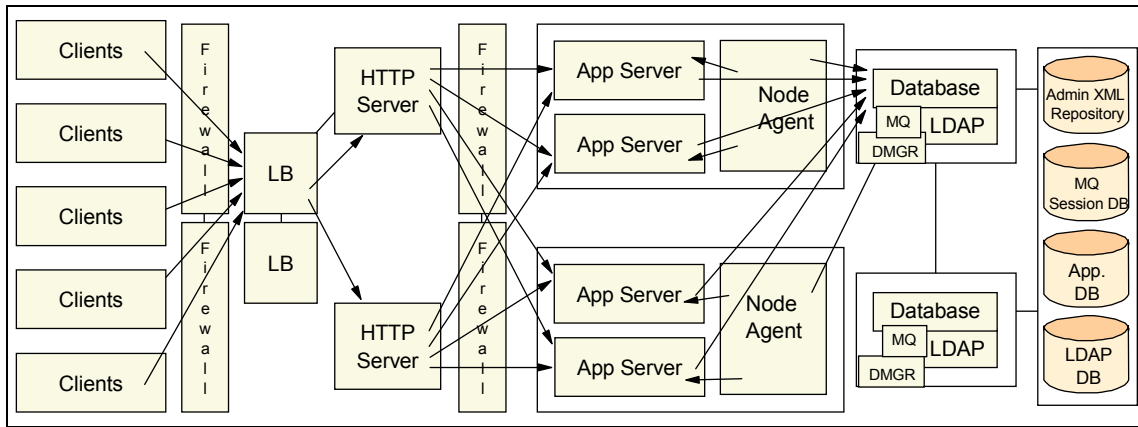


Figure 8-6 WebSphere system HA level 4

## WebSphere system HA level 5

For WebSphere system HA level 5, two data centers are used for disaster recovery. As shown in Figure 8-7 on page 332, one data center is in the primary service site, and the other data center is a backup. Geographically remote mirroring is used to replicate data between two sites in an asynchronous way. Therefore, both sites can be far away. When one site goes down, the backup site can serve requests after an interruption of up to a few hours.

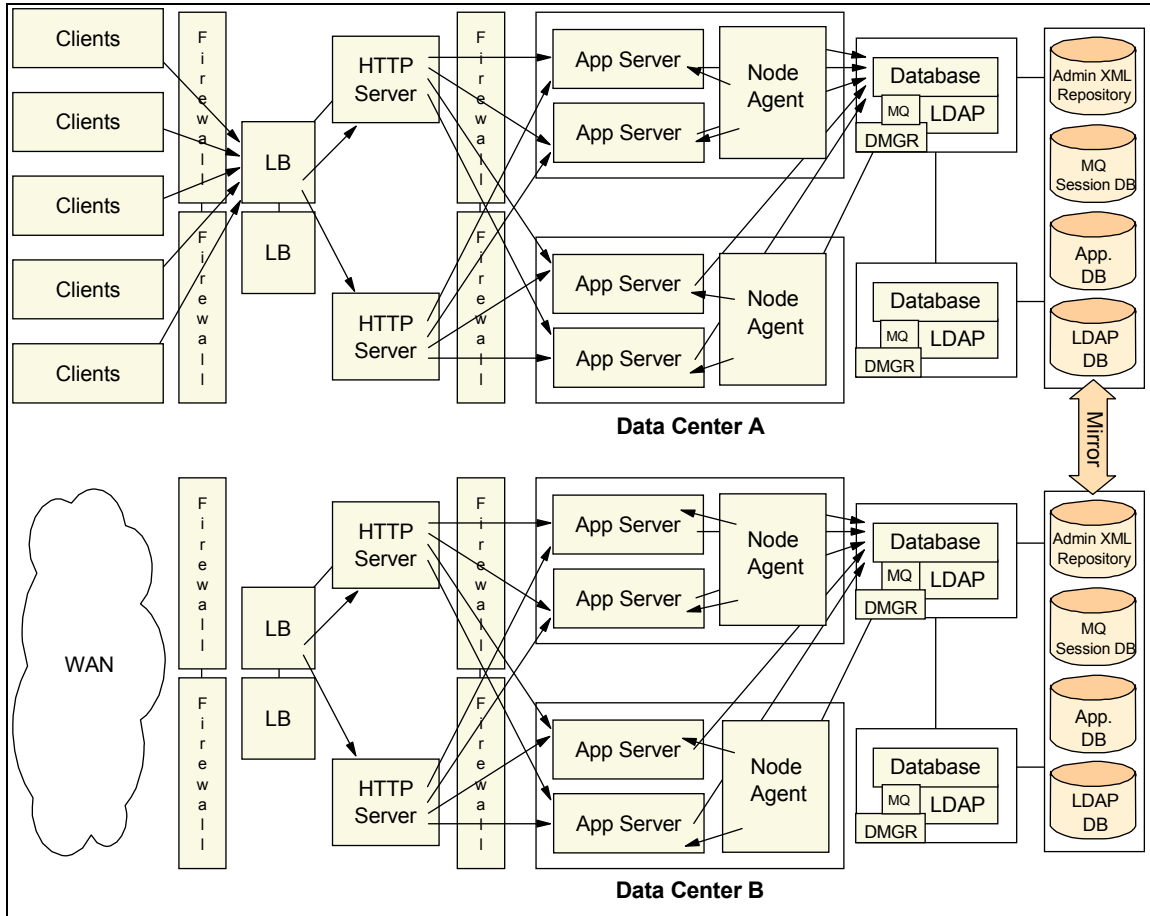


Figure 8-7 WebSphere system HA level 5

## WebSphere system HA level 6

For WebSphere system HA level 6, as shown in Figure 8-8 on page 333, operations will not stop because of a failure in any component at any site. Geographical clustering is used to synchronize data between two sites. Clients can access both sites at the same time; therefore, scalability and performance are also enhanced. When one data center fails, the other data center will pick up all of the traffic. However, since both sites need to be synchronized in real time, the two sites cannot be far away because of communication latency.

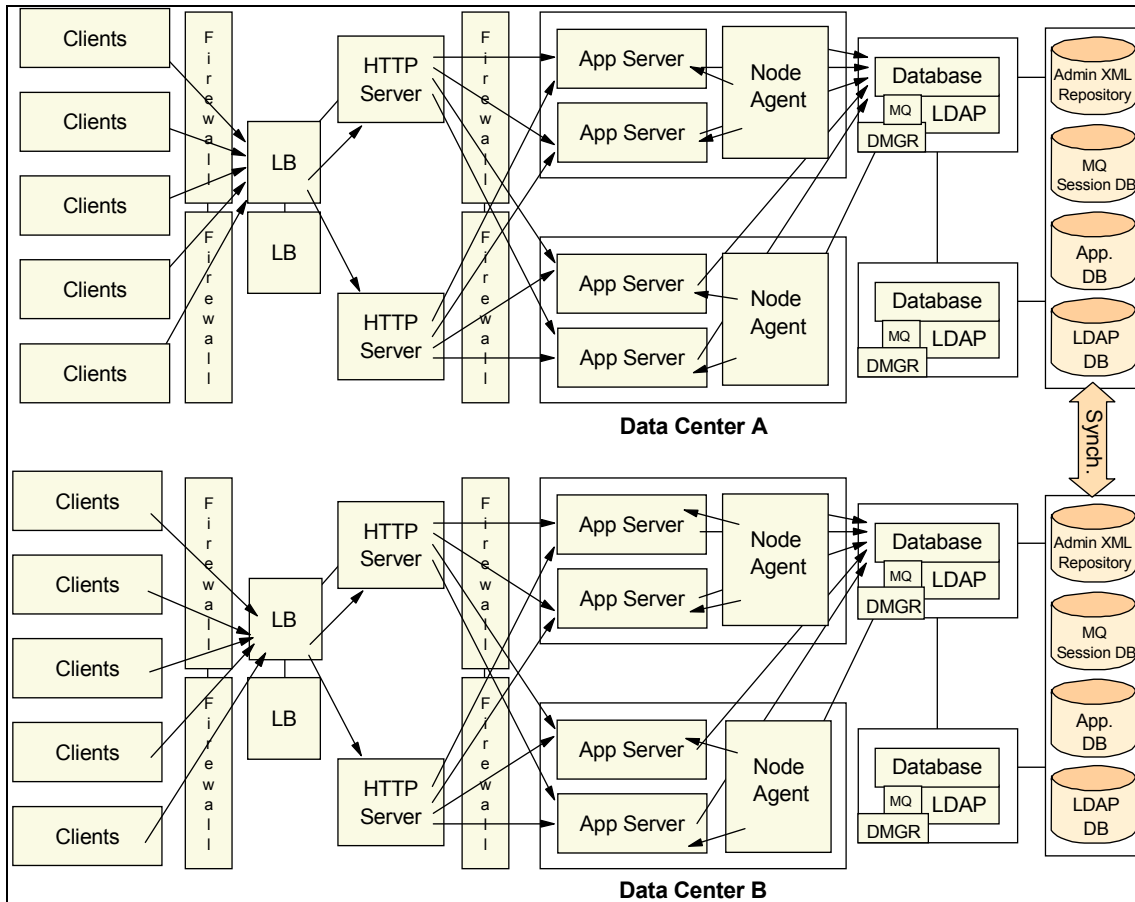
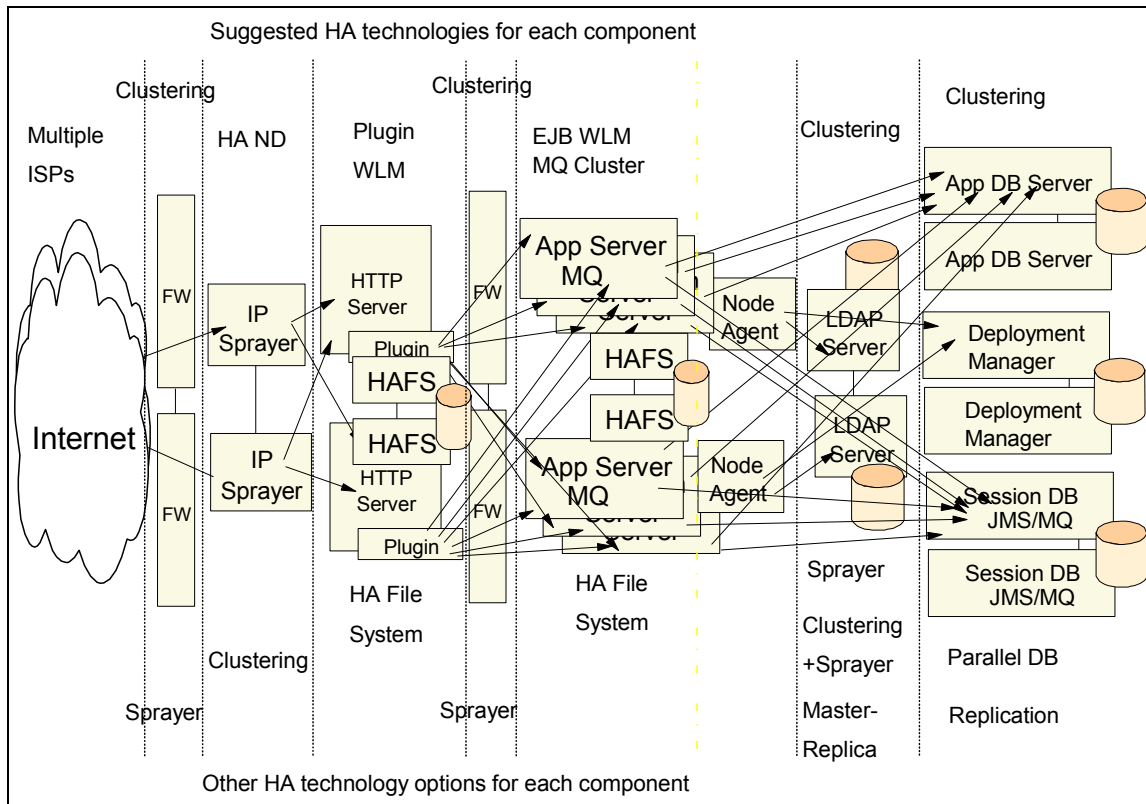


Figure 8-8 WebSphere system HA level 6

In summary, we have many choices regarding HA technologies and/or HA topologies for the end-to-end WebSphere system, as shown in Figure 8-9 on page 334 for HA technologies and in Figure 8-10 on page 335 for HA best possible topologies.



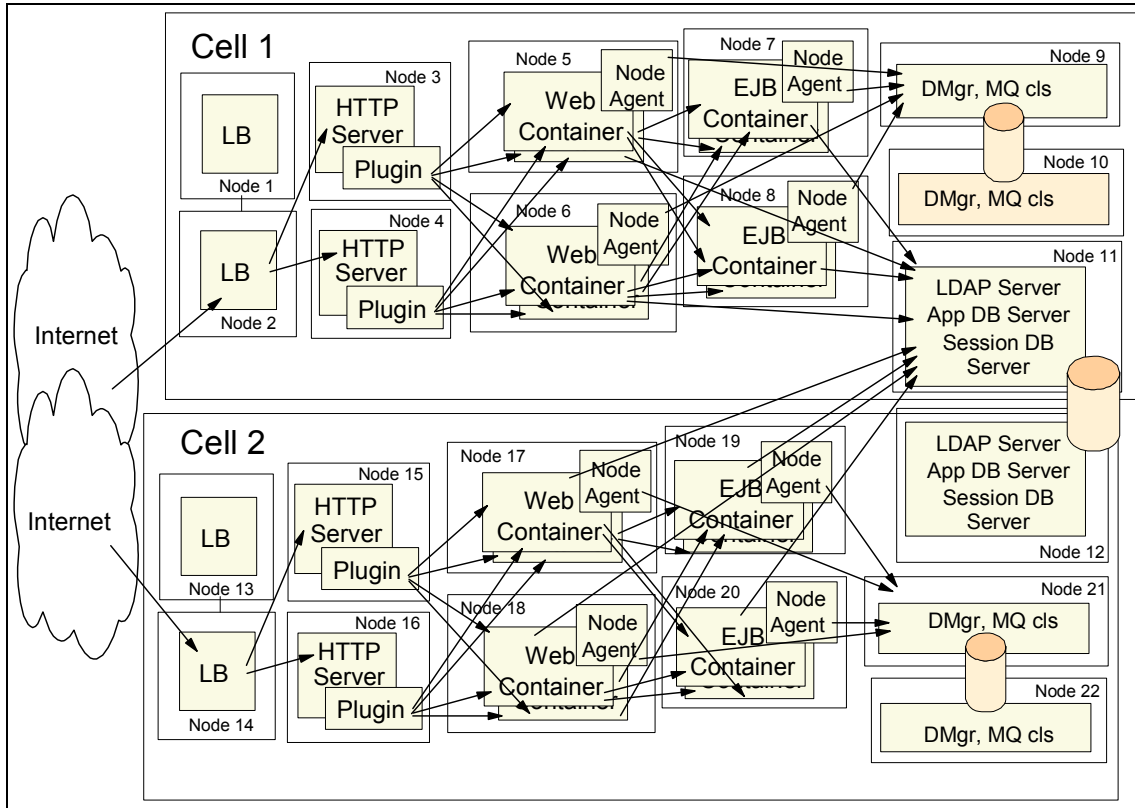


Figure 8-10 Two WebSphere administrative domains HA topology

Multiple WebSphere administrative domains (cells) provide not only the capacity for migration and upgrades of WebSphere, application software, and hardware without interruption of WebSphere services (7x24 upgrades/migration), but also make the deployment of different versions of WebSphere possible (WebSphere V4.x and V5.x are not supported in the same domain).

Multiple smaller domains may also provide better performance than a single large domain (cell), because there is less interprocess communication in a smaller domain. However, multiple WebSphere administrative domains add substantially more maintenance effort, since administration must be performed on each domain. Scripts together with a network file system can reduce daily administration efforts for multiple WebSphere administrative domains.

### 8.3.6 Planning and evaluating your WebSphere HA solutions

We recommend that you use the following steps when planning and evaluating your end-to-end WebSphere HA solutions:

1. As we discussed before, analyze your requirements:
  - a. Do you need continuous operations? Most customers do not need continuous operations; therefore, upgrades of software/hardware can be performed offline.
  - b. What kind of availability do you need during your hours of operation?
  - c. What is the performance requirement during a failover? For example, the performance may be impacted because fewer nodes/servers serve client requests after some of the nodes/servers fail.
2. Analyze the cost factors. How much will you lose when your system is unavailable, and how much can you invest in your system availability? If downtime costs you more, you should invest more to improve your system availability.
3. Estimate the setup and administrative complexity. More complicated systems require skilled people and more configuration and administration effort.
4. Consider all the components in an end-to-end WebSphere system. Usually, the overall availability is dominated by the weakest point of the end-to-end WebSphere system chain. Consider the possibility of failure in each component and its failover time.
5. Analyze failover time, which mainly includes fault-detection time and recovery time. For different failover mechanisms/techniques, the failover time is different.
6. Analyze the recovery point, where your processing resumes after a failover. It is directly related to the amount of work that is lost during a failover.
7. Understand the programming models. This concerns whether the failover is transparent to clients and whether one component's failover is transparent to other components in an end-to-end WebSphere system. Some services are able to perform failover transparently, since the failover is masked from the application. Others have options for adding application code that can retry when necessary.
8. Finally, know that there is usually more than one solution to address a given failure. Some solutions may have special restrictions. Analyze the trade-offs between the different solutions.



## 8.4 Failover terms and mechanisms

People use the same term *failover* for different failovers and different failover mechanisms. We distinguish these different failover mechanisms here.

An object or an application includes two distinct aspects: functions and data. Therefore, we have process availability and data availability. If a function is not associated with individual data or states, it is easy to achieve high availability by simply restarting this function process once the old process crashes. However, the reality is that functions are associated with individual data or state, some with persisted data in database or files, such as Entity EJBs. We need to make data management systems highly available to all processes and ensure data integrity since the failed process may damage data.

*Failover* refers to the single process that moves from the primary system to the backup system in the cluster. The failure recovery includes several steps:

1. Stop and exit the failed process.
2. Release the resources.
3. Detach the disk array.
4. Reattach the disk array to the backup system.
5. Check the disk and file system.
6. Repair the data integrity.
7. Gain all resources for running the process in the backup system.
8. Start the process in the backup system.

This failover process takes several minutes after the fault is detected. This approach can be used for both function-centric or data-centric applications for both active/standby and active/active configurations.

Figure 8-11 on page 338 and Figure 8-12 on page 338 show active/standby failover configurations.

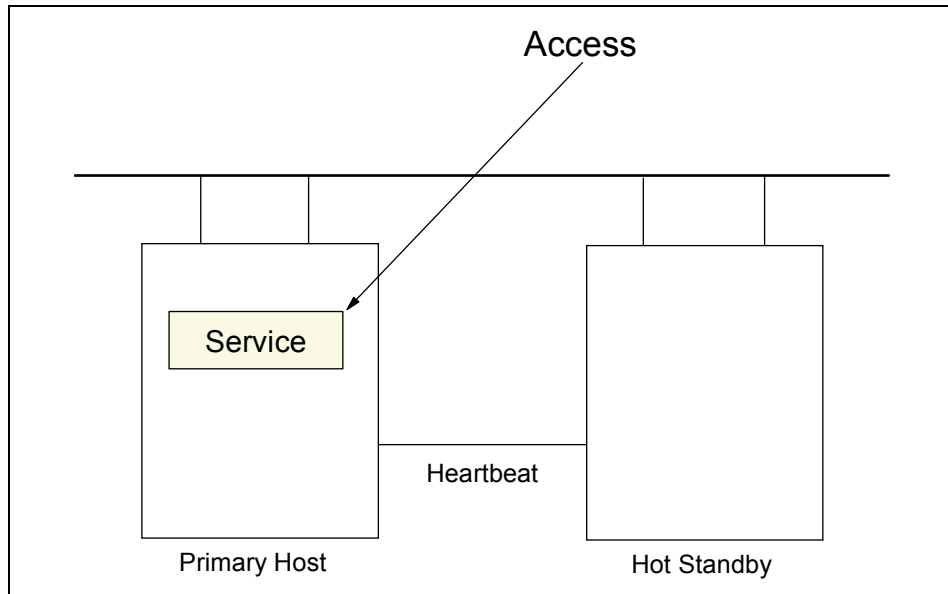


Figure 8-11 Active/standby configuration before failover

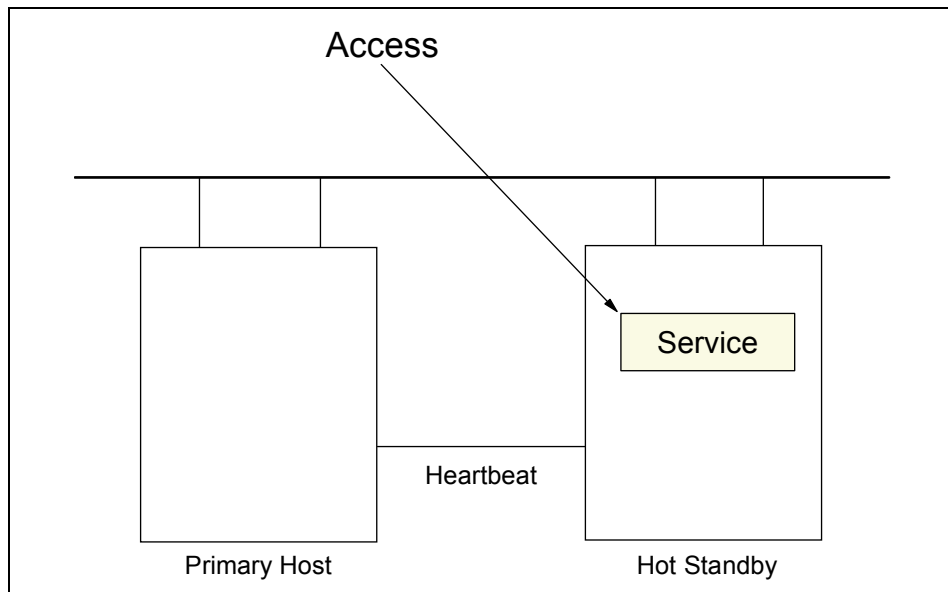
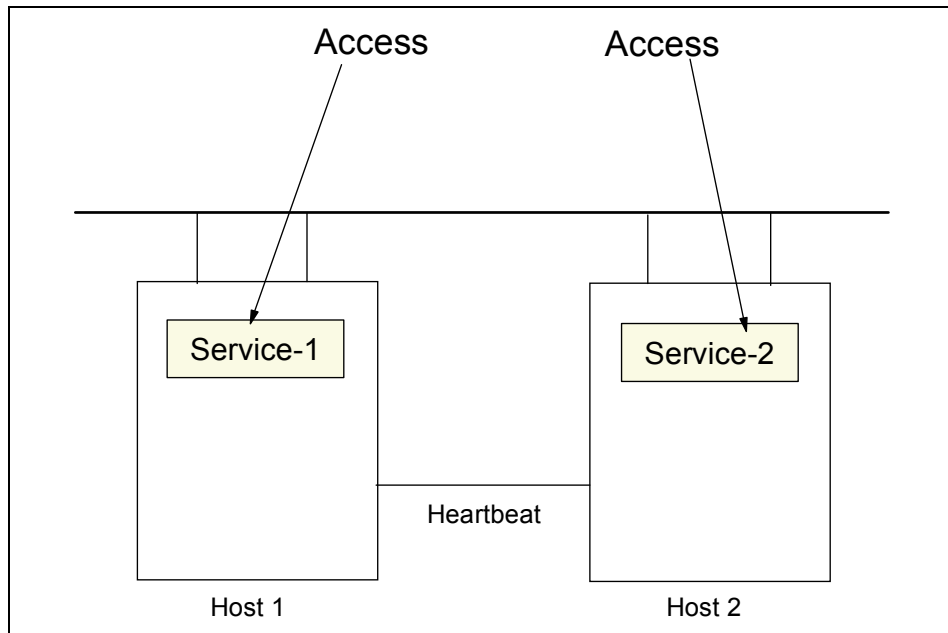


Figure 8-12 Active/standby configuration after failover

Figure 8-13 and Figure 8-14 on page 340 show active/active failover configurations.



*Figure 8-13 Active/active configuration before failover*

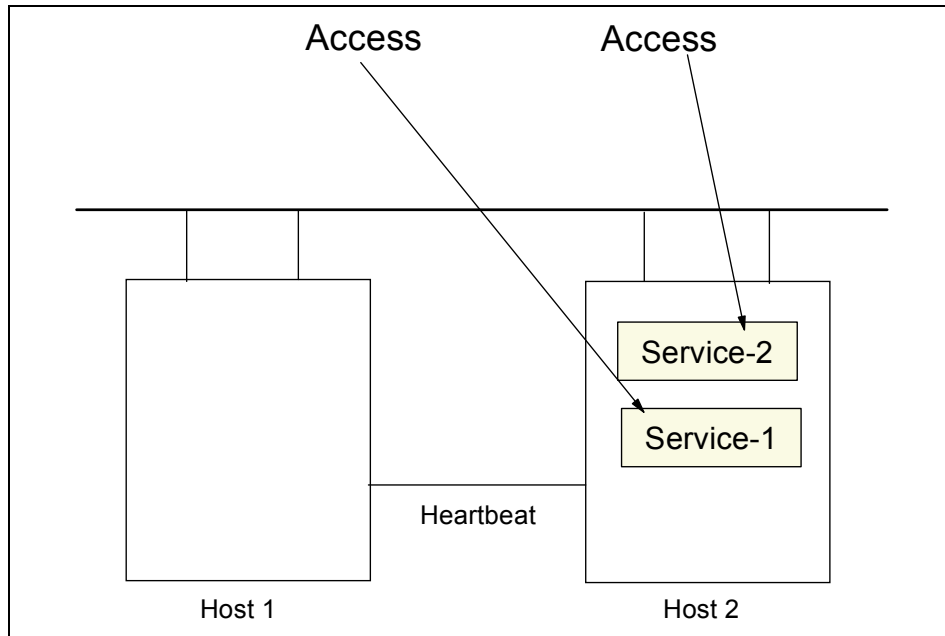


Figure 8-14 Active/active configuration after failover

*Fail back* or *fallback* is similar to failover, but occurs from the backup system to the primary system once the primary system is back online. For mutual takeover, since the backup node has its original application running, as shown in Figure 8-14, failing back will improve the performance of both applications, as shown in Figure 8-13 on page 339.

**Note:** The two processes in the mutual takeover hosts are different applications. You should distinguish them from the processes that belong to the same application (we will discuss this later).

*Fail fast* refers to *uncoordinated* process pairs: the backup process is pre-existent, as shown in Figure 8-15 on page 341. Please note the following points:

- ▶ These processes belong to the *same* application.
- ▶ These processes are *not coordinated*; in other words, these processes do not know each other's running state.
- ▶ These processes are pre-existent on hosts, which is different from the failover case where a new process starts with the takeover of resources after the original process fails.

- These processes do not take resources from each other.

Fast fail cannot be used in data-centric applications. It relies on 1-n mapping (such as sprayer) to handle client traffic and process errors. Tuning the connection and TCP/IP timeout parameters is a key part of this kind of failover performance. We do need to balance normal running performance and failover performance. Too short a connection and TCP/IP timeout may improve the failover performance, but harm the normal running performance.

WebSphere uses this approach for EJB workload management and plug-in workload management, where cluster layer resources are not shared among processes, processes are uncoordinated and do not know each other's state. This approach is suitable only for service-centric applications; it cannot be used for data-centric applications. We need another HA data management approach for Entity EJBs data and other data.

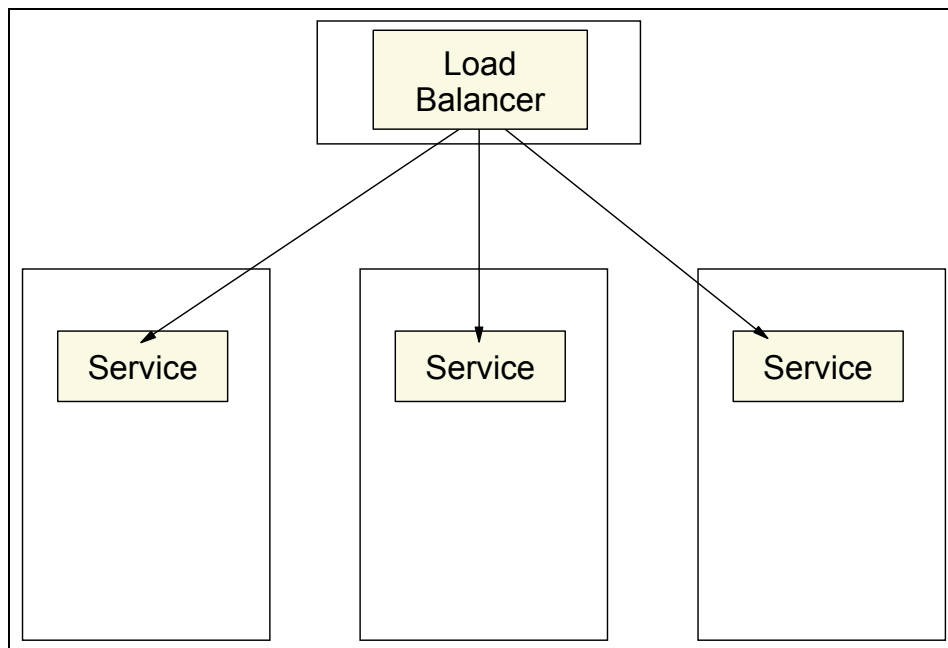


Figure 8-15 Fail fast without clustering

A feedback mechanism can be applied to the system to dynamically balance workload across the service nodes, as shown in Figure 8-16 on page 342.

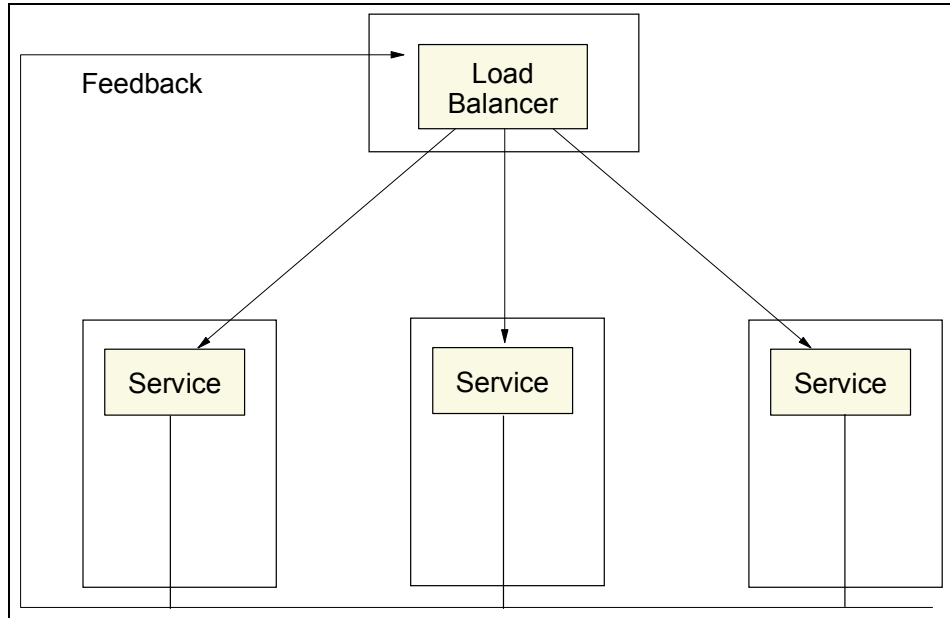


Figure 8-16 Fail fast with feedback for workload balancing

*Fail transparent* is defined as coordinated process pairs: we have a primary process and a backup process running on separate processors, as shown in Figure 8-17 on page 343. The primary process sends checkpoints to the backup process. If the primary process fails, the backup process takes over. This mechanism is based on the backup process anticipating the failure of the primary process and then taking over without affecting the work in progress or user sessions. This approach requires that each process know the states of other processes as well as the state of the environment. Therefore, a cluster management layer is required and the application must query the status from this cluster layer. In other words, the processes are coordinated, and errors are handled transparently. Oracle Parallel Server/TAF is an example of the successful implementation of this kind of failover.

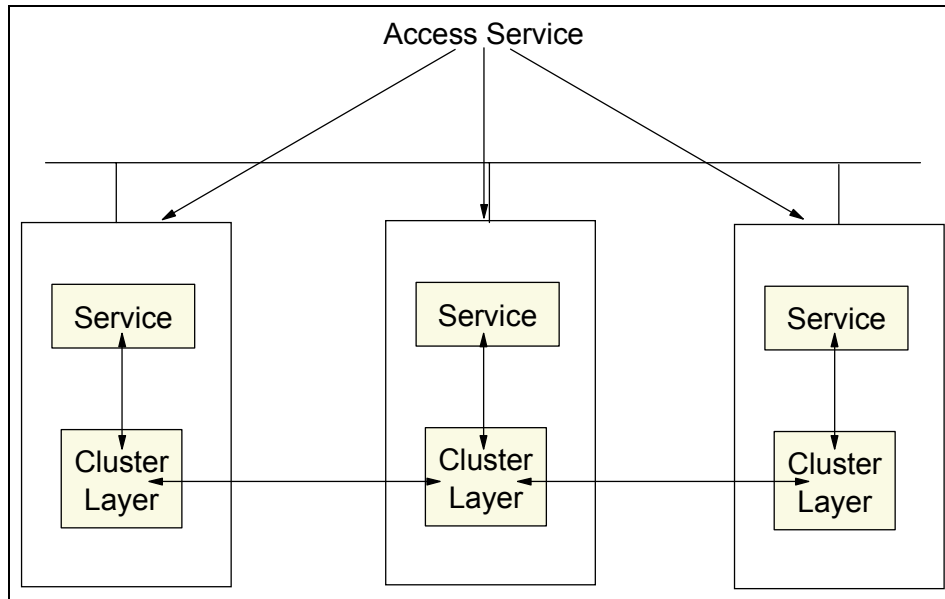


Figure 8-17 Fail transparent with clustering

In summary, we have three approaches to achieve high availability:

- ▶ Process and dependent resource group takeover

This approach requires cluster information and can be used for both data-centric and service-centric applications. Its implementation is difficult.

- ▶ Multiple uncoordinated processes


This approach does not require cluster information. However, this approach cannot be used for data-centric applications since it does not support the resource group concept.

- ▶ Multiple coordinated processes

This approach requires cluster information, and its implementation is very difficult. It can be used for both data-centric and service-centric applications. It can achieve transparent failover without interruption.







## WebSphere Application Server failover and recovery

IBM WebSphere Application Server Network Deployment V5.1 failover and recovery is realized through the WebSphere Workload Management (WLM) mechanism. In addition, other clustering software such as HACMP, VERITAS Cluster Server, MC/ServiceGuard, Sun Cluster Server, and Microsoft Cluster Service can also be used to enhance application server failover and recovery, especially because of aspects such as the two-phase transaction failover and transaction log recovery and the automatic recovery of failed hosts where WebSphere Application Servers are running. We discuss all of these techniques.

For this chapter, we assume that you are using the IBM WebSphere Application Server Network Deployment V5.1 WLM functionality.

This chapter should be used in conjunction with Chapter 5, “Plug-in workload management and failover” on page 133 and Chapter 6, “EJB workload management” on page 215.

## 9.1 Overview

WebSphere Application Server V5 provides support for creating an application server cluster using a server template. This concept is different from the server group in WebSphere Application Server V4; the server template or model is not active in Version 5, and a change to the model will not impact cluster servers *after* creation. Server templates can contain a Web container, an EJB container, or both. From a server template, a WebSphere administrator can create any number of application server instances, or cluster members. These cluster members can all reside on a single node (systems) or can be distributed across multiple nodes in the WebSphere domain or WebSphere cell. Cluster members can be administered as a single unit by manipulating the server cluster object. Cluster members can share application workload and provide failover support. All servers can be manually divided into primary servers and backup servers. If one of the primary servers fails, work can continue to be handled by other primary servers in the server cluster, as long as primary servers are still available. If all the primary servers are unavailable, the request will fail over to the first backup server. If the first backup server fails, the next backup server will be used.

You may have thin Web clients or/and thick Java/C++ clients. When using clustered WebSphere Application Servers, your clients can be redirected either automatically or manually (depending on the nature of the failure) to another healthy server in the case of a failure of a clustered application server.

When an HTTP request reaches the HTTP server, a decision must be made. Some requests for static content may be handled by the HTTP server. Requests for dynamic content or some static content will be passed to a Web container running in a WebSphere Application Server. Whether the request should be handled or passed to WebSphere is decided by the WebSphere Web server plug-in, which runs in-process with the HTTP server. For these WebSphere requests, high availability for the Web container becomes an important piece of the failover solution.

When an EJB client makes calls from the Web container or client container or from outside, the request is handled by the EJB container in one of the clustered application servers. If that server fails, the client request is redirected to another available server.

If you don't workload manage your WebSphere Application Servers and you don't use any other clustering software, your system will not provide any failover support. All of your Web or Java clients will fail if your WebSphere Application Server fails.

If you workload manage your WebSphere cluster servers, your system provides satisfactory failover support for most cases. In addition, when using other

clustering software such as HACMP, Sun Cluster Server, VERITAS Cluster Server, MC/ServiceGuard, and Microsoft Cluster Service, you also can provide the automatic failover and recovery of two-phase transactions and hosts.

## 9.2 Web container clustering and failover

As mentioned before, a WebSphere environment may include several HTTP server instances. Each HTTP server is configured to run the WebSphere HTTP plug-in in its process. Each request coming into the Web server is passed through this plug-in, which uses its configuration information to determine if the request should be routed to WebSphere, and if so, which Web container the request should be routed to (see Figure 9-1 on page 348). These Web containers may be running on the same machine as the HTTP server, a different machine, or a combination of the two.

In WebSphere V5, several new functions have been added to enhance WebSphere plug-in failover and recovery:

- ▶ Server weight: Minimizes unavailability due to overloading (refer to Chapter 5, “Plug-in workload management and failover” on page 133 for more information on server weights).
- ▶ Ordered cluster server failover: Eliminates the problem that two cluster servers may be selected as the session failover target concurrently.
- ▶ Primary server and backup server lists: Provides one more level of failover support.
- ▶ Non-blocking connection: Reduces the impact of operating system's TCP/IP keep-alive timeout.
- ▶ Memory-to-memory session replication: Enhances session data availability.

The Web container failover support in WebSphere V5 is provided by three mechanisms:

- ▶ WebSphere Web container server cluster, which creates server process redundancy for failover support.
- ▶ The workload management routing technique built into the WebSphere Web server plug-in. It controls the routing of client requests among redundant server processes.
- ▶ Session management and failover mechanism, which provides HTTP session data for redundant server processes.

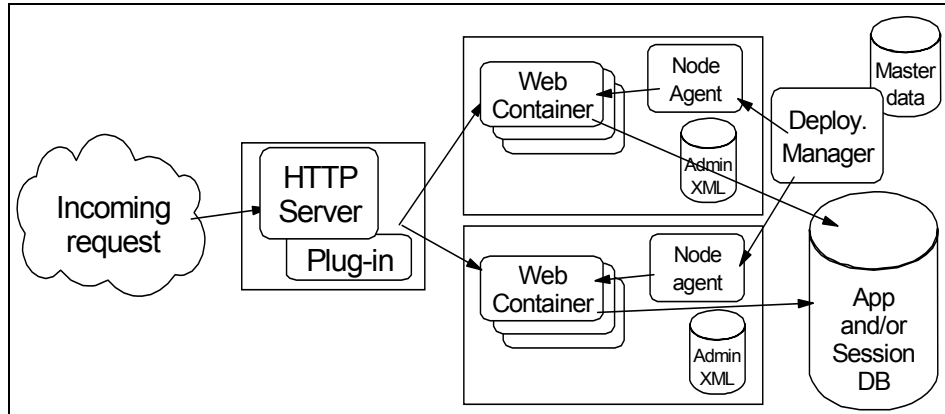


Figure 9-1 WebSphere Application Server (Web container) high availability

## 9.2.1 Web container failures and failover

When a Web container fails, it is the responsibility of the HTTP server plug-in to detect this failure and mark the Web container unavailable. Web container failures are detected based on the TCP response values, or lack of response, to a plug-in request. There are five types of failover scenarios for Web containers:

- ▶ Expected server process failures, for example after stopping the server.
- ▶ Unexpected server process failures, for example the server JVM crashes.
- ▶ Server network problems, for example the network cable is disconnected or a router is broken.
- ▶ Unexpected and expected system problems, for example a system shutdown, operating system crashes, or power failures.
- ▶ Overloading of Web clients, for example a denial of service attack, system is too small to handle a large number of clients, or server weight is inappropriate.

In the first two cases, the physical machine where the Web container is supposed to be running will still be available, although the Web container port will not be available. When the plug-in attempts to connect to the Web container port to process a request for a Web resource, the machine will refuse the connection, causing the plug-in to mark the application server as down.

In the third and fourth events, however, the physical machine is no longer available to provide any kind of response. In these events, if non-blocking connection is not enabled, the plug-in will wait for the local operating system to time out the request before marking the application server unavailable. While the plug-in is waiting for this connection to time out, requests routed to the failed

application server appear to hang. The default value for the TCP timeout varies based on the operating system. While these values can be modified at the operating system level, adjustments should be made with great care. Modifications may result in unintended consequences in both WebSphere and other network dependent applications running on the machine. This problem can be eliminated by enabling non-blocking connection. Refer to “ConnectTimeout setting” on page 207 for more information.

In the fifth case, client overloading can make a healthy server unavailable and cause a server overloading failover. This is explained in 9.2.4, “Stream and overloading failover” on page 356.

## 9.2.2 Web server plug-in failover performance tuning

As explained earlier, the Web server plug-in uses an XML configuration file called plugin-cfg.xml to determine information about the WebSphere domain it is serving.

The plug-in also uses this configuration file to determine how to route HTTP requests to the Web containers. For detailed information about how the plug-in makes this determination, refer to 5.4, “WebSphere plug-in workload management” on page 144.

You can modify the plugin-cfg.xml file for better failover performance:

- ▶ Divide the servers into a primary server list and a backup server list. This is a new feature in WebSphere V5, also called two-level failover support. When the plugin-cfg.xml file is generated, by default all servers are in the primary server list. You can manually move some servers into the backup server list. Refer to “Primary and backup servers” on page 163 and “Primary and backup servers cluster two-level failover” on page 349 for details.
- ▶ Adjust the plugin-cfg.xml refresh time (for details, see “Refresh interval of the plug-in” on page 152).
- ▶ Adjust the RetryInterval. See “RetryInterval and operating system TCP timeout” on page 352 for more information.
- ▶ Add ConnectTimeout attribute for each server (this is explained in “ConnectTimeout setting” on page 207).

### Primary and backup servers cluster two-level failover

WebSphere V5 supplies a two-level failover functionality. When the plugin-cfg.xml is generated, all servers are initially listed under the PrimaryServers tag. You can manually move selected servers to be listed under the BackupServers tag, as shown in Example 9-1 on page 350.

#### Example 9-1 Primary and Backup Server lists in plugin-cfg.xml

---

```
<PrimaryServers>
  <Server Name="HAClusterServer1"/>
  <Server Name="HAClusterServer3"/>
</PrimaryServers>
<BackupServers>
  <Server Name="HAClusterServer2"/>
  <Server Name="HAClusterServer4"/>
</BackupServers>
```

---

The Web server plug-in will not route requests to any server in the Backup Server list as long as there are application servers available from the Primary Server list. Within the PrimaryServers, the plug-in routes traffic according to server weight and/or session affinity. When all servers in the Primary Server list are unavailable, the plug-in will route traffic to the first available server in the Backup Server list. If the first server in the Backup Server list is not available, the request will be routed to the next server in the Backup Server list. Weighted round robin routing is *not* performed for the servers in the Backup Server list. The plug-in just uses one server at a time. Refer to “Primary and backup servers” on page 163 for more details on how the plug-in performs the server selection.

This two-level failover mechanism provides more advanced failover support, allowing better use of available server capacity.

Typical two-level failover behavior is shown in the trace in Example 9-2.

#### Example 9-2 Two-level failover trace

---

```
TRACE: ws_server_group: serverGroupAddPrimaryServerName: adding
WebHAbbMember1 to primary server list
TRACE: ws_server_group: serverGroupAddPrimaryServerName: adding
WebHAbbMember3 to primary server list
TRACE: ws_server_group: serverGroupAddBackupServerName: adding
WebHAbbMember2 to backup server list
TRACE: ws_server_group: serverGroupAddBackupServerName: adding
WebHAbbMember4 to backup server list
TRACE: ws_server_group: serverGroupGetFirstServer: getting the first server
TRACE: ws_server_group: serverGroupGetNextServer: getting the next server
Only primary servers participate weighted Round Robin:
TRACE: ws_server_group: serverGroupNextRoundRobinServer: Round Robin load
balancing
TRACE: ws_server_group: serverGroupNextRoundRobinServer: numPrimaryServers
is 2
TRACE: ws_server_group: assureWeightsValid: group WebHACluster
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first
primary server
TRACE: ws_server_group: sum_wlb_weights: WebHAbbMember1: 1 max, 0 cur.
```

```

TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next
primary server
TRACE: ws_server_group: sum_wlb_weights: WebHAbbMember3: 1 max, 0 cur.
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next
primary server
TRACE: ws_server_group: sum_wlb_weights := 0
TRACE: ws_server_group: serverGroupGetFirstServer: getting the first server
TRACE: ws_server_group: serverGroupGetNextServer: getting the next server
TRACE: ws_server_group: serverGroupGetNextServer: getting the next server
TRACE: ws_server_group: serverGroupGetNextServer: getting the next server
TRACE: ws_server_group: serverGroupGetNextServer: getting the next server
-----Retry another one
TRACE: ws_server_group: serverGroupCheckServerStatus: Checking status of
WebHAbbMember3, ignoreWeights 0, markedDown 1, retryNow 1, wlbAllows 1
TRACE: ws_server_group: serverGroupCheckServerStatus: Time to retry server
WebHAbbMember3
TRACE: ws_server_group: lockedServerGroupUseServer: Server WebHAbbMember3
picked, weight 0.
TRACE: ws_common: websphereFindTransport: Finding the transport
TRACE: ws_common: websphereFindTransport: Setting the transport:
10.55.90.84 on port 9081
TRACE: ws_common: websphereExecute: Executing the transaction with the app
server
TRACE: ws_common: websphereGetStream: Getting the stream to the app server
TRACE: ws_transport: transportStreamDequeue: Checking for existing stream
from the queue
ERROR: ws_common: websphereGetStream: Failed to connect to app server, OS
err=10061
ERROR: ws_common: websphereExecute: Failed to create the stream
ERROR: ws_server: serverSetFailoverStatus: Marking WebHAbbMember3 down
ERROR: ws_common: websphereWriteRequestReadResponse: Failed to execute the
transaction to 'WebHAbbMember3'; will try another one
TRACE: ws_server_group: serverGroupNextRoundRobinServer: Round Robin load
balancing
TRACE: ws_server_group: serverGroupNextRoundRobinServer: numPrimaryServers
is 2
TRACE: ws_server_group: assureWeightsValid: group WebHACluster
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first
-----Get backup Server
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first
primary server
TRACE: ws_server_group: serverGroupCheckServerStatus: Checking status of
WebHAbbMember1, ignoreWeights 0, markedDown 1, retryNow 0, wlbAllows 1
TRACE: ws_server_group: serverGroupCheckServerStatus: Server WebHAbbMember1
is marked down; retry in 60
ERROR: ws_server_group: serverGroupNextRoundRobinServer: Failed to find a
server; all could be down
TRACE: ws_server_group: serverGroupGetNextUpBackupServer: Getting the next
up backup server

```

```
TRACE: ws_server_group: serverGroupGetFirstBackupServer: getting the first
backup server
TRACE: ws_server_group: serverGroupCheckServerStatus: Checking status of
WebHAbbMember2, ignoreWeights 1, markedDown 0, retryNow 0, wlbAllows 1
TRACE: ws_common: websphereFindTransport: Finding the transport
TRACE: ws_common: websphereFindTransport: Setting the transport:
10.55.90.22 on port 9082
TRACE: ws_common: websphereExecute: Executing the transaction with the app
server
TRACE: ws_common: websphereGetStream: Getting the stream to the app server
TRACE: ws_transport: transportStreamDequeue: Checking for existing stream
from the queue
```

---

## RetryInterval and operating system TCP timeout

If a request to an application server in a cluster fails, and there are other application servers in the group, the plug-in will transparently reroute the failed request to the next application server in the routing algorithm. The unresponsive application server is marked unavailable and all new requests will be routed to the other application servers in the server cluster.

The amount of time the application server remains unavailable after a failure is configured by the `RetryInterval` property on the `<ServerGroup>` attribute. If this attribute is not present, the default value is 60 seconds.

```
<ServerCluster Name="HACluster"> RetryInterval=600>
```

The failover behavior is shown in Example 9-3.

### *Example 9-3 Marked down application server*

---

```
serverGroupCheckServerStatus: Server WebHAbbMember4 is marked down; retry
in 598
serverGroupCheckServerStatus: Checking status of WebHAbbMember4,
ignoreWeights 0, markedDown 1, retryNow 0, wlbAllows 1
...
serverGroupCheckServerStatus: Server WebHAbbMember4 is marked down; retry
in 579
serverGroupCheckServerStatus: Checking status of WebHAbbMember4,
ignoreWeights 0, markedDown 1, retryNow 0, wlbAllows 1
...
serverGroupCheckServerStatus: Server WebHAbbMember4 is marked down; retry
in 6
serverGroupCheckServerStatus: Checking status of WebHAbbMember4,
ignoreWeights 0, markedDown 1, retryNow 0, wlbAllows 1
...
serverGroupCheckServerStatus: Server WebHAbbMember4 is marked down; retry
in 0
```



```
serverGroupCheckServerStatus: Checking status of WebHAbbMember4,  
ignoreWeights 0, markedDown 1, retryNow 1, wlbAllows 1
```

---

When the `RetryInterval` expires, the plug-in will add the application server back into the routing algorithm and attempt to send a request to it. If the request fails or times out, the application server is again marked unavailable for the length of the `RetryInterval`.

The proper setting for the `RetryInterval` will depend on your environment, particularly the value of the operating system TCP timeout value and how many application servers are configured in the cluster. Setting the `RetryInterval` to a small value will allow an application server that becomes available to quickly begin serving requests. However, too small of a value can cause serious performance degradation, or even cause your plug-in to appear to stop serving requests, particularly in a machine outage situation.

To explain how this can happen, let's look at an example configuration with two machines, which we will call A and B. Each of these machines is running two clustered application servers (CM1 and CM2 on A, CM3 and CM4 on B). The HTTP server and plug-in are running on an AIX system with a TCP timeout of 75 seconds, the `RetryInterval` is set to 60 seconds, and the routing algorithm is weighted round robin. If machine A fails, either expectedly or unexpectedly, the following process occurs when a request comes in to the plug-in:

1. The plug-in accepts the request from the HTTP server and determines the server cluster.
2. The plug-in determines that the request should be routed to cluster member CM1 on system A.
3. The plug-in attempts to connect to CM1 on machine A. Because the physical machine is down, the plug-in waits 75 seconds for the operating system TCP timeout interval before determining that CM1 is unavailable.
4. The plug-in attempts to route the same request to the next cluster member in its routing algorithm, CM2 on machine A. Because machine A is still down, the plug-in must again wait 75 seconds for the operating system TCP timeout interval before determining that CM2 is also unavailable.
5. The plug-in attempts to route the same request to the next cluster member in its routing algorithm, CM3 on system B. This application server successfully returns a response to the client, over 150 seconds after the request was first submitted.
6. While the plug-in was waiting for the response from CM2 on system A, the 60-second `RetryInterval` for CM1 on system A expired, and the cluster member is added back into the routing algorithm. A new request will soon be

routed to this cluster member, which is still unavailable, and this lengthy waiting process will begin again.

To avoid this problem, we recommend setting a more conservative `RetryInterval`, related to the number of cluster members in your configuration. A good starting point is 10 seconds + (`#_of_cluster_members * TCP_Timeout`). This ensures that the plug-in does not get stuck in a situation of constantly trying to route requests to the failed members. In the scenario described before, this setting would cause the two cluster members on system B to exclusively service requests for 235 seconds before the cluster members on system A are retried, resulting in another 150-second wait.

As mentioned earlier, another option is to configure your application servers to use a non-blocking connection. This eliminates the impact of the operating system TCP/IP timeout. “ConnectTimeout setting” on page 207 explains how to configure this option.

### 9.2.3 Network failures

When the network becomes unavailable, the Node Agent will force its managed application servers to stop if the loopback is configured for 127.0.0.1. If the network becomes available within 10 minutes (which is the timeout value for the Node Agent), the Node Agent can restart the stopped servers. If the network is unavailable beyond the 10 minutes downtime, the application servers will remain down and you will need to manually start the application servers. The behavior after disconnecting the network cable for a long time is shown in Example 9-4.

#### *Example 9-4 Network failure Node Agent trace*

---

```
NodeSyncTask  A ADMS0003I: Configuration synchronization completed
successfully.
NodeSync      E ADMS0015E: The synchronization request can not be completed
because the node agent can not communicate with the deployment manager.
NodeSyncTask  A ADMS0016I: Configuration synchronization failed.
NodeAgent     W ADML0063W: Cannot contact server "WebHAbbMember4". Force to
stop this server if it is still running.
NodeAgent     A ADML0064I: Restarting unreachable server "WebHAbbMember4".
NodeAgent     W ADML0063W: Cannot contact server "WebHAbbMember3". Force to
stop this server if it is still running.
NodeAgent     A ADML0064I: Restarting unreachable server "WebHAbbMember3".
NodeSync      E ADMS0015E: The synchronization request can not be completed
because the node agent can not communicate with the deployment manager.
NodeSyncTask  A ADMS0016I: Configuration synchronization failed.
NodeAgent     W ADML0040E: Timed out waiting for server "WebHAbbMember4"
initialization: 600 seconds
NodeAgent     W ADML0040E: Timed out waiting for server "WebHAbbMember3"
initialization: 600 seconds
```

```
NodeSync      E ADMS0015E: The synchronization request can not be completed
because the node agent can not communicate with the deployment manager.
NodeSyncTask  A ADMS0016I: Configuration synchronization failed.
NodeSyncTask  A ADMS0003I: Configuration synchronization completed
successfully.
```

---

Example 9-5 shows the server trace.

*Example 9-5 Network failure Server trace*

---

```
WsServer      E WSVR0003E: Server WebHAbbMember3 failed to start
com.ibm.ejs.EJSEException: Could not register with Location Service Daemon;
nested exception is:
  org.omg.CORBA.TRANSPORT: Host unreachable:
connect:host=10.55.90.84,port=9900  minor code: 4942F301  completed: No
org.omg.CORBA.TRANSPORT: Host unreachable:
connect:host=10.55.90.84,port=9900  minor code: 4942F301  completed: No
at
com.ibm.CORBA.transport.TransportConnectionBase.connect(TransportConnection
Base.java:338)
[10:10:27:674 CDT] 68cf7b9a WsServer      E WSVR0009E: Error occurred
during startup
```

---

To resolve this problem, configure a loopback alias to a systems' real IP address, not the default loopback of 127.0.0.1.

In AIX, use the following command:

```
ifconfig lo0 alias my_ip netmask 255.255.255.0
```

After a short-time network outage, the Node Agent can restart servers automatically as shown in Example 9-6.

*Example 9-6 Automatic restart of application servers after network failure*

---

```
:NodeSyncTask  A ADMS0003I: Configuration synchronization completed
successfully.
NodeAgent      A ADML0000I: Server initialization completed. Process id is:
45848
DiscoveryMBea  I ADMD0023I: Process discovered (name: WebHAbbMember4, type:
ManagedProcess, pid: 45848)
NodeSyncTask  A ADMS0003I: Configuration synchronization completed
successfully.
NodeAgent      A ADML0000I: Server initialization completed. Process id is:
116004
DiscoveryMBea  I ADMD0023I: Process discovered (name: WebHAbbMember3, type:
ManagedProcess, pid: 116004)
```

## 9.2.4 Stream and overloading failover

WebSphere Application Server V5.1 is designed to handle a large number of client requests concurrently. However, sometimes overloading a server can cause client requests to fail even though the server is really healthy. It is difficult to catch overloading failures because of the differences between a test environment and a production system. Thus, you need to carefully test your application with the expected load. In our lab testing, we used Mercury LoadRunner and AKstress (from the Web Performance Tools) to simulate various client loads. The Web Performance Tools are no longer available. Refer to 19.1, “Testing the performance of an application” on page 822 for information about other available load testing tools.

WebSphere V5 provides an embedded HTTP transport in its Web container that is connected to the external HTTP server through the plug-in. An open connection between the Web server and the Web container is called a *stream*. If there are more connections than available threads, the connections start to backlog, waiting for free threads. If the maximum number of backlog connections is reached, new connections will be refused. The plug-in will treat this healthy server as a failed server, and mark it as down. This leads to an overloading failover.

There are several parameters you can tune to reduce overloading failovers:

- ▶ Connection backlog
- ▶ Maximum keep-alive connections
- ▶ Maximum requests per keep-alive connection
- ▶ Keep-alive timeout
- ▶ I/O timeout
- ▶ Minimum thread size
- ▶ Maximum thread size
- ▶ Thread inactivity timeout

Refer to 19.4, “Performance tuning guidelines” on page 843 for more information about these parameters, and refer to the WebSphere InfoCenter, located at:

<http://www.ibm.com/software/webservers/appserv/infocenter.html>

You may also need to adjust your server weight when an overloading failover occurs.

## 9.3 HTTP session failover

HTTP session objects can be used within a Web application to maintain information about a client across multiple HTTP requests. For example, on an online shopping Web site the Web application needs to maintain information about what each client has placed in its shopping cart. The session information is stored on the server, and a unique identifier for the session is sent back to the client as a cookie or through the URL rewriting mechanism.

### Server ID

When the server is created, WebSphere assigns a unique CloneID for each server, the cloneID is shown in the plugin-cfg.xml file:

```
<ServerCluster Name="HACluster">
  <Server CloneID="u307p2vq" LoadBalanceWeight="2"
Name="HAClusterServer1"></Server>
  <Server CloneID="u307p48r" LoadBalanceWeight="3"
Name="HAClusterServer2"></Server>
  <Server CloneID="u307p62u" LoadBalanceWeight="4"
Name="HAClusterServer3"></Server>
  <Server CloneID="u307p71m" LoadBalanceWeight="5"
Name="HAClusterServer4"></Server>
</ServerCluster>
```

The CloneID is used to create the serverGroup object. The plug-in will bind the CloneID and a client's sessionID when it receives the client's first request as shown in Example 9-7.

#### *Example 9-7 Binding CloneID and sessionID*

---

```
TRACE: ws_server_group: serverGroupCreate: Creating the server group object
TRACE: ws_server_group: serverGroupSetName: Setting the name: HACluster
TRACE: ws_server: serverCreate: Creating the server object
TRACE: ws_server: serverSetName: Setting name HAClusterServer1
TRACE: ws_server: serverSetCloneID: Setting clone id u307p2vq
TRACE: ws_transport: transportCreate: Creating transport
TRACE: ws_transport: transportSetProtocol: Setting the protocol http
TRACE: ws_transport: transportSetPort: Setting the port 9081
TRACE: ws_transport: transportSetHostname: Setting the hostname 9.5.90.22
TRACE: ws_server: serverAddTransport: Adding transport 9.5.90.22
TRACE: ws_server_group: serverGroupAddServer: Adding server
HAClusterServer1
TRACE: ws_server_group: serverGroupAddServer: Server HAClusterServer1 has
session affinity
TRACE: ws_server: serverCreate: Creating the server object
TRACE: ws_server: serverSetName: Setting name HAClusterServer2
TRACE: ws_server: serverSetCloneID: Setting clone id u307p48r
TRACE: ws_transport: transportCreate: Creating transport
```

```
TRACE: ws_transport: transportSetProtocol: Setting the protocol http
TRACE: ws_transport: transportSetPort: Setting the port 9082
TRACE: ws_transport: transportSetHostname: Setting the hostname 9.5.90.22
TRACE: ws_server: serverAddTransport: Adding transport 9.5.90.22
TRACE: ws_server_group: serverGroupAddServer: Adding server
HAClusterServer2
TRACE: ws_server_group: serverGroupAddServer: Server HAClusterServer2 has
session affinity
TRACE: ws_server: serverCreate: Creating the server object
TRACE: ws_server: serverSetName: Setting name HAClusterServer3
TRACE: ws_server: serverSetCloneID: Setting clone id u307p62u
TRACE: ws_transport: transportCreate: Creating transport
TRACE: ws_transport: transportSetProtocol: Setting the protocol http
TRACE: ws_transport: transportSetPort: Setting the port 9081
TRACE: ws_transport: transportSetHostname: Setting the hostname 9.5.90.84
TRACE: ws_server: serverAddTransport: Adding transport 9.5.90.84
TRACE: ws_server_group: serverGroupAddServer: Adding server
HAClusterServer3
TRACE: ws_server_group: serverGroupAddServer: Server HAClusterServer3 has
session affinity
TRACE: ws_server: serverCreate: Creating the server object
TRACE: ws_server: serverSetName: Setting name HAClusterServer4
TRACE: ws_server: serverSetCloneID: Setting clone id u307p71m
TRACE: ws_transport: transportCreate: Creating transport
TRACE: ws_transport: transportSetProtocol: Setting the protocol http
TRACE: ws_transport: transportSetPort: Setting the port 9082
TRACE: ws_transport: transportSetHostname: Setting the hostname 9.5.90.84
TRACE: ws_server: serverAddTransport: Adding transport 9.5.90.84
TRACE: ws_server_group: serverGroupAddServer: Adding server
HAClusterServer4
TRACE: ws_server_group: serverGroupAddServer: Server HAClusterServer4 has
session affinity
```

---

## Session ID

The HTTP protocol itself is stateless and it cannot carry stateful information for subsequent requests by itself. WebSphere V5 can work around this HTTP protocol defect and provide session support as well as session failover support. WebSphere V5 keeps the user's session information on the server, using different options:

- ▶ In-memory, which provides no failover support.
- ▶ Into a database or into message brokers, which provides failover support.

WebSphere recognizes the user's session information in three ways:

- ▶ SSL session ID
- ▶ Cookies
- ▶ URL rewriting

Refer to 5.5, “Session management” on page 166 for more information.

## Ordered routing process

To achieve session affinity, the plug-in will parse the client's session information, extract the previous CloneID, and match this CloneID with the CloneIDs in the server cluster object. When the matching cluster member is found, the server will retrieve this client's session from the in-memory cache. When the matched application server is not available, the plug-in will route the client to another server, and appends the new server's CloneID into the client's session. WebSphere V5 uses the weight-based ordered routing, which eliminates the possibility that two different servers are selected after a failover for concurrent (multiple thread) clients (which might have been a problem in WebSphere V4.x where the adoptive server selection was random).

### *Example 9-8 Trace ordered routing process*

---

```
TRACE: ws_common: websphereParseCloneID: Returning list of clone ids
TRACE: ws_server_group: serverGroupFindClone: Looking for clone
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first
primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID
'u307p2vq' to server clone id 'u307p2vq'
TRACE: ws_server_group: serverGroupFindClone: Match for clone
'HAClusterServer1'
TRACE: ws_server_group: serverGroupCheckServerStatus: Checking status of
HAClusterServer1, ignoreWeights 1, markedDown 1, retryNow 0, wlbAllows 0
TRACE: ws_server_group: serverGroupCheckServerStatus: Server
HAClusterServer1 is marked down; retry in 58
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next
primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID
'u307p2vq' to server clone id 'u307p62u'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next
primary server
TRACE: ws_server_group: serverGroupGetFirstPrimaryServer: getting the first
primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID
'u307p62u' to server clone id 'u307p2vq'
TRACE: ws_server_group: serverGroupGetNextPrimaryServer: getting the next
primary server
TRACE: ws_server_group: serverGroupFindClone: Comparing curCloneID
'u307p62u' to server clone id 'u307p62u'
TRACE: ws_server_group: serverGroupFindClone: Match for clone
'HAClusterServer3'
TRACE: ws_server_group: serverGroupCheckServerStatus: Checking status of
HAClusterServer3, ignoreWeights 1, markedDown 0, retryNow 0, wlbAllows 1
TRACE: ws_server_group: lockedServerGroupUseServer: Server HAClusterServer3
picked, weight 3.
```

```
TRACE: ws_common: websphereHandleSessionAffinity: Setting server to
HAClusterServer3
TRACE: ws_common: websphereFindTransport: Finding the transport
TRACE: ws_common: websphereFindTransport: Setting the transport: 9.5.90.84
on port 9081
TRACE: ws_common: websphereExecute: Executing the transaction with the app
server
TRACE: ws_common: websphereGetStream: Getting the stream to the app server
TRACE: ws_transport: transportStreamDequeue: Checking for existing stream
from the queue
TRACE: ws_common: websphereSocketIsClosed: Checking to see if socket is
still open
TRACE: ws_common: websphereGetStream: Using existing stream from transport
queue
TRACE: lib_htrequest: htrequestWrite: Writing the request:
```

---

### 9.3.1 Session affinity and failover

WLM works different with session affinity. We will discuss the following options:

- ▶ No session support needed
- ▶ No session information in the client
- ▶ Session affinity without session persistence
- ▶ Session persistence and failover

#### No session support needed

By default, session affinity is enabled in WebSphere. If none of your applications needs session support, you can remove the CloneID in the plugin-cfg.xml file to make your request processing faster. Doing so eliminates the time-consuming comparison process:

```
<ServerCluster Name="HACluster">
  <Server CloneID="u307p2vq" LoadBalanceWeight="2"
Name="HAClusterServer1"></Server>
  <Server CloneID="u307p48r" LoadBalanceWeight="3"
Name="HAClusterServer2"></Server>
  <Server CloneID="u307p62u" LoadBalanceWeight="4"
Name="HAClusterServer3"></Server>
  <Server CloneID="u307p71m" LoadBalanceWeight="5"
Name="HAClusterServer4"></Server>
</ServerCluster>
```

However, if any application in your application servers is stateful, you will lose all state information during a failover. Remember that this setting is for the whole server cluster, not for a single Web application. It is not necessary to disable the server session support for session-less clients.



## No session information in the client

When server session affinity is enabled, the plug-in will look for session information in each incoming request. If the plug-in fails to find any session information from the incoming client, the plug-in will assume that no session affinity is needed for this client. The plug-in will ignore session affinity and route this client using weighted round robin as shown in the trace in Example 9-9.

*Example 9-9 Trace: No session information in the client*

---

```
TRACE: ws_common: websphereHandleSessionAffinity: Checking for session affinity
TRACE: ws_common: websphereHandleSessionAffinity: Checking the SSL session id
TRACE: lib_htrequest: htrequestGetCookie: Looking for cookie: 'SSLJSESSION'
TRACE: lib_htrequest: htrequestGetCookie: No cookie found for: 'SSLJSESSION'
TRACE: ws_common: websphereParseSessionID: Parsing session id from '/snoop'
TRACE: ws_common: websphereParseSessionID: Failed to parse session id
TRACE: ws_common: websphereHandleSessionAffinity: Checking the app server
session id
TRACE: lib_htrequest: htrequestGetCookie: Looking for cookie: 'JSESSIONID'
TRACE: lib_htrequest: htrequestGetCookie: No cookie found for: 'JSESSIONID'
TRACE: ws_common: websphereParseSessionID: Parsing session id from '/snoop'
TRACE: ws_common: websphereParseSessionID: Failed to parse session id
```

---

## Session affinity without session persistence

When server session support is enabled and an incoming client request has session information, HTTP sessions are kept in the memory of the application server containing the Web application. The plug-in will route multiple requests for the same HTTP session to the same server by examining the CloneID information that is stored with the session key. Thus, the session is kept for a client during the following requests.

The plug-in will route the client to an alternative server in case the original server is not available. If the alternative server cannot retrieve the session information, all session information is lost and the client needs to start over again. The failover process fails, but the client can continue without the previous session information.

## Session persistence and failover

In order to make a session fail over successfully, you need to persist the session and to make the session information available in the adoptive server. WebSphere V5 adds the new feature of memory-to-memory session replication, in addition to database session persistence. When session affinity and session persistence are both enabled, HTTP sessions are held in-memory of the application server containing the Web application, and are also periodically persisted to the database or published to the broker(s). The plug-in will route multiple requests for the same HTTP session to the same server. This server can then retrieve the

information from its in-memory session cache. If this server is unavailable, the request is routed to another server, which reads the session information from a database or receives the session information from the broker locally (if available) or goes one more hop to retrieve the session information from a remote broker. The current WebSphere V5 HTTP WLM routing has no information about which servers have session information locally.

Another new and nice feature is that in WebSphere V5, the session manager can be configured at various levels:

- ▶ Web module level
- ▶ Enterprise application level

All Web modules inside the enterprise application inherit the configuration defined at this level.

- ▶ Application server Web container level

All Web modules in the server's Web container have the same configuration defined at this level. This is the default in WebSphere V5; WebSphere V4 supports only this level.

Therefore, you can have different persistent mechanisms in the Web containers at the same time to tune failover for each Web module. How much session information might be lost depends on the frequency of the session persistence, which is configurable on the application server from the Administrative Console. This is discussed in the following section.

### 9.3.2 Session update methods and failover session data loss

The WebSphere V5 session manager drives session replication and persistence. WebSphere V5 has three methods to trigger the session data persistence, each of which has different failover impacts. The fewer the updates and the longer the update interval, the better the server performance, but the more session data is lost in case of a failover.

#### **At the end of servlet service**

When the write frequency is set to the *end of servlet service*, WebSphere will write/publish the session into database/replication stores at the completion of the `HTTPServlet.service()` method call. All new session data between two calls is lost after a failover. The adoptive server will retrieve the session data from the last `HTTPServlet.service()` method call.

#### **Manually sync() in the code using the IBM extension**

IBM extended the Servlet Specification to add the functionality of letting an application decide when to update its session data. The application needs to

invoke the `sync()` method explicitly to write/publish the session into database/replication stores. All new session data since the last invoke of `sync()` is lost after a failover.

### Time-based (at a specified time interval)

The session is written/published into database/replication stores at the preset time interval. All new session data since the last update is lost during a failover.

WebSphere V5 offers several choices for setting the time interval using the Administrative Console as shown in Figure 9-2.

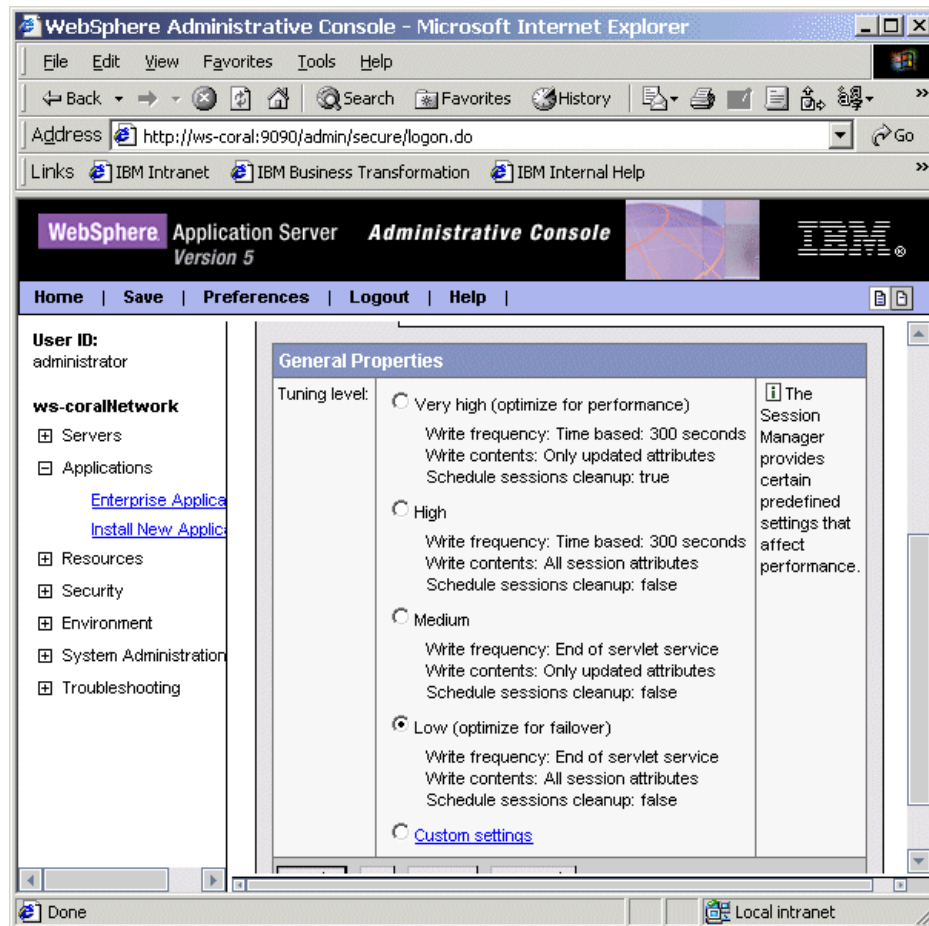


Figure 9-2 WebSphere HTTP session failover optimization

You can also use custom settings to fine tune these parameters.

### 9.3.3 Session persistence and failover

WebSphere V5 offers three different methods to persist sessions:

- ▶ In-memory
- ▶ Persistent to database
- ▶ Memory-to-memory

#### In-memory

In-memory session management provides the best performance for HTTP requests as long as the user's requests are always directed to the same server and this server never fails.

The local in-memory session manager doesn't persist the session in the host. Thus, when the host becomes unavailable, all session information is lost and when the server becomes available again, it cannot recover the previous session.

The local in-memory session manager doesn't share session information with other servers in the cluster. Thus, if the server fails, the adoptive server (another server in the cluster) cannot retrieve the session information, and the failover will fail.

#### Persistent to database

If session data is persisted to a database and the application server fails, another server can retrieve the session from the database. Therefore, the client request can continue to be processed. The session data loss amount depends on the time difference between the last update and the failure time, with the maximum being the update interval (see 9.3.2, "Session update methods and failover session data loss" on page 362).

If the database service is not available, the adoptive server cannot retrieve the session from the database. The WebSphere session manager will retry for a while and the client appears to hang. If the database is still not available, the failover process will fail. Therefore, the session database is an SPOF. We discuss how to enable database high availability in Chapter 12, "WebSphere data management high availability" on page 435.

#### Memory-to-memory

WebSphere V5 provides a third mechanism for session management: memory-to-memory replication using WebSphere Data Replication Service (DRS). There are three topological models for memory-to-memory replication:

- ▶ Buddy system with single replica
- ▶ Dedicated replication server
- ▶ Peer-to-peer replication (default)

If the original server fails, the HTTP WLM plug-in will route the user's request to another server in the server cluster (adoptive server). The session manager in the adoptive server will look whether there is a copy of the user's session locally. If not, it will go to another store to retrieve this user's session.

Considerations for configuring the replication topology are:

1. At least two replication stores are needed for high availability in case a single replicated copy is also unavailable
2. The message complexity of session replication should be minimized

While the buddy system and the dedicated replication server may have fewer replication messages, they are also prone to failures if many servers fail at the same time. The peer-to-peer or client/server models provide more resistance to multiple server failures. Channel partitioning provided with WebSphere V5 can be used in an environment with a large number of servers to reduce the replication message complexity.

The HTTP WLM plug-in doesn't know which servers have the replicated session locally. It simply routes the user's request to an adoptive server in the cluster, according to server weights, without any session store consideration. With the peer-to-peer model, the session information is distributed across all application servers, that is, each application server retrieves sessions from other application servers, and each application server provides sessions to other application servers. Therefore, no matter which server the HTTP plug-in routes the user's request to, it can always get the session locally.

In order to make dynamic Web contents highly available, you also need to make LDAP, database, and file system highly available. This is discussed in detail in Chapter 12, "WebSphere data management high availability" on page 435 and Chapter 13, "High availability of LDAP, NFS, and firewall" on page 503.

## 9.4 EJB container failover

Many J2EE applications rely on Enterprise JavaBeans (EJBs) to provide business logic. EJB clients can be servlets, JSPs, J2EE client, stand-alone Java applications, or even other EJBs. The approach for EJB failover in WebSphere 5 is to have server redundancy and smart routing techniques, subject to the constraints of resource availability, data integrity, and transaction and process affinity.

The following issues are related to EJB container failover and recovery:

1. EJB client redundancy - how many copies of the same client?
  - Clients in the same application server (servlets, JSPs, EJBs)

- Clients in a different application server (servlets, JSPs, EJBs)
  - Stand-alone Java clients
  - J2EE clients
  - Other clients (C++, third-party ORB, others)
2. EJB container server redundancy for failover support.
  3. Fault isolation and data integrity.
  4. Resource redundancy (EJB datastore, JMS resource, LDAP).

We discuss these issues in the following sections.

### 9.4.1 EJB client redundancy and bootstrap failover support

The first thing for any EJB client is to look up the home of the bean (except MDB). Two kinds of EJB clients need to be considered:

- Multiple copies of the same EJB client may exist in a client request chain.  
For example, Web clients from Web containers that are workload managed, EJB clients from another EJB container server cluster, or EJB clients from their own server cluster. In this case, EJB clients can use their own server to bootstrap with the default provider URL. If the bootstrap fails, the EJB client fails. This failure should be handled by the previous server, for example the HTTP plug-in. Another version of the same client in a different container may bootstrap from its server successfully. By using client redundancy, EJB failover and high availability is achieved.
- Only one copy of the same client.  
For example, a Java client, J2EE client, C++ client, or third-party ORB client. In this case, if the client is bootstrapped to only one server, it will fail if that server fails since the client is not redundant. You need to bootstrap this client to as many bootstrap servers as possible.

```
prop.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
prop.put(Context.PROVIDER_URL, "corbaloc::host1:9810,:host2:9810");
Context initialContext = new InitialContext(prop);
try { java.lang.Object myHome = initialContext.lookup("MyEJB");
```

In addition to the IIOP URL, WebSphere V5 also supports the CORBA object URL as specified in J2EE 1.3. It is very convenient to use multiple bootstrap addresses for automatic retries. Every WebSphere server process contains a naming service, and you can bootstrap to any combination of servers. It is good practice to bootstrap to the servers in a cluster since the initialContext is workload managed and you can use the simple name in the lookup. Otherwise, you need to specify the fully qualified names in the lookup and InitialContext in

the Node Agents is not workload managed (so it has no failover capability), although they use the default port of 2809, which seems convenient.

For J2EE clients, you can specify a bootstrap host and port in the launch command line, and try different hosts/ports until you succeed if you use the default URL provider.

Once you look up the EJB home, the naming server will return an indirect IOR, LSD, and routing table, and the WLM plug-in will redirect the client to one of many clustered EJB containers.

## 9.4.2 EJB container redundancy and EJB WLM failover support

High availability of the EJB container is achieved using a combination of the WebSphere server cluster support and the Workload Management (WLM) plug-in to the WebSphere ORB. As discussed earlier, WebSphere server clusters allow multiple instances of an application server to be created from a template. These multiple application servers, or cluster members, have a single administration point and the ability to share workload.

### Redundant server topology for EJB failover support

As shown in Figure 9-3, horizontal and vertical scaling is used to have application server redundancy to tolerate possible process and machine failures.

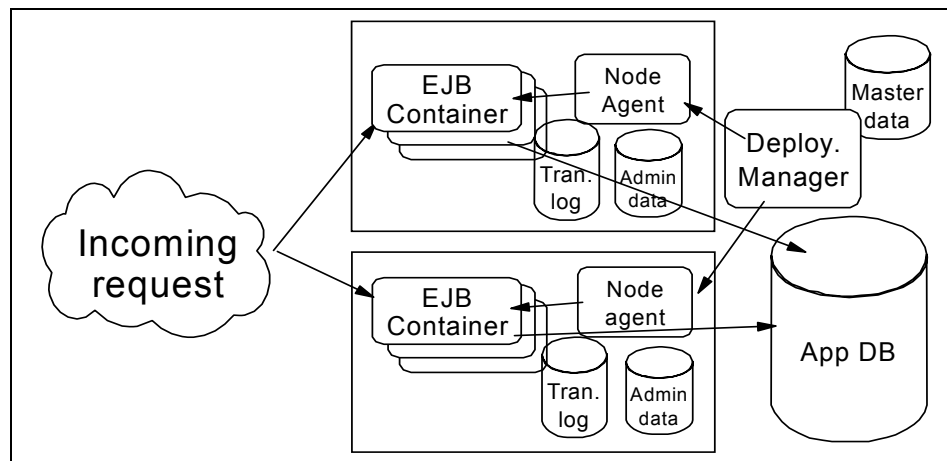


Figure 9-3 WebSphere EJB container failover

We discuss how to fail over two-phase transactions and recover tranlog in 9.5, “Enhancing WebSphere HA using clustering software” on page 376 and how to fail over a database in Chapter 12, “WebSphere data management high availability” on page 435.

The mechanisms for routing workload managed EJB requests to multiple cluster members are handled on the client side of the application. In WebSphere V5.0, this functionality is supplied by a workload management plug-in to the client ORB and the routing table in LSD hosted in the Node Agent. The WLM failover support for EJBs is to maintain the routing table and to modify the client ORB to redirect traffic in case of a server failure.

There are the following possible failures:

- ▶ Expected server process failures, for example stopping the server.
- ▶ Unexpected server process failures, for example the server JVM crashes.
- ▶ Server network problems, for example a network cable is disconnected or a router is broken.
- ▶ Unexpected and expected machine problems, for example a system shutdown, operating system crashes, or power failures.
- ▶ Overloading of EJB clients, for example a denial of service attack, the system is too small to handle a large number of clients, or the server weight is inappropriate.

### 9.4.3 EJB WLM routing

The implementation of the routing pattern for IIOP requests to an application server is a combination of the two client routing patterns: client and client with advisor. This implementation is to support both those clients with the distributed platform WebSphere code and also those clients that support the IIOP interface without the WebSphere extensions.

It is expected that the non-WebSphere distributed clients will consist of WebSphere for z/OS®, CORBA C++, or non-IBM ORBs. The WebSphere distributed clients are cluster aware and have WLM support built into the client.

**Note:** We are using the terms “cluster aware” and “cluster unaware” clients here. However, when looking at a trace, such as shown in Example 9-7 on page 357, you will see the term “serverGroup” instead of “cluster”.

This support has knowledge of the servers in the cluster and implements the routing policy resulting in the client making the routing decision and directing the request to a specific server. The non-WebSphere distributed clients are cluster unaware clients and do not have any WLM routing or cluster knowledge and must be given a specific server to direct the client's request to. In this case we employ a Location Service Daemon (LSD), which acts as an advisor to a client, directing incoming requests to the appropriate server.



Within the LSD approach, a client obtains an indirect IOR (possibly from a lookup in JNDI, or as a return argument) which contains routing information to the LSD rather than to the application server. The client request is then sent to the LSD to determine a direct IOR to be used for the object request. As part of the resolution of the IOR, some WLM logic is added to the LSD. The LSD's ORB will contain hooks that will be used in the WLM code to determine how to handle the request. The following sections describe the different functions based on the client type.

### ***Cluster aware***

If the client contains the distributed WebSphere WLM code and has yet to obtain the initial routing information, the initial request will flow to the LSD. The WLM director code in the LSD will determine the correct server and return the direct IOR to the client. This direct IOR will then be used to route the request to the specific server. However, since this client contains the distributed WebSphere WLM code, the request to the server will contain WLM service context data, and the WLM code on the server will look at this context data to determine whether the client has back-level cluster information. If required, the WLM code on the server adds cluster information into the service context list on the response back to the client. The WLM code on the client then updates its cluster information (if newer data was available). From this point on, the WLM code on the client has updated cluster information. Thus for all subsequent requests the client has the capability to choose the specific server to handle the request. This client-based routing is basically how our current Java EJB clients work in WebSphere today.

### ***Cluster unaware***

If the client does not have the distributed WLM code active, the LSD will invoke the WLM director where it is the director's responsibility to handle all of the WLM policy and affinity decisions that need to be made. The client does not have any knowledge of the cluster, so the LSD must determine the specific server the request will be sent to. This direct IOR will then be returned to the client and be used to invoke all requests on the object. What this means is that every request the client makes on this object will go to the server that the LSD determined. This is different from cluster aware clients, which will determine what server to use for every request. For cluster unaware clients, a load balancing mechanism would be added to the server that would detect when it was getting overloaded and would stop a request by returning an error to the client. This error would trigger the client to return to the LSD again to determine another direct IOR to use. In addition, if a server being used by a cluster unaware client goes down or is disconnected, an error will be returned and the client will return to the LSD and start the process over.

### ***Location Service Daemon (LSD)***

If only one LSD is configured for each cluster, the potential for a single point of failure exists. To prevent the single point of failure, a set of LSDs per cluster will be configured and if an LSD fails, one of the other LSDs in the group will be used.

## Routing algorithm

While the weighted round robin algorithm described in Chapter 5, “Plug-in workload management and failover” on page 133 is the main routing algorithm, there are situations that will override how the request is routed. These situations are described in the next sections.

### *In-process*

If a client request is to an object that is already in the same process as the client, the request will automatically be routed to the in-process object. In this case, no weights are decremented. The in-process optimization overrides all other selection criteria.

### *Prefer local*

"Prefer local" is an additional attribute that can be selected on a given server cluster. This attribute will instruct the selection logic to keep all client requests local to the same node as the client if possible. Thus, if the Prefer local setting is enabled, and a request from a client can be serviced by an application server that is local to the same machine as the client, the request will be routed to that local server. Prefer local still decrements the weight for each request, and if multiple application servers (cluster members) are local to the client, the standard weighted round robin algorithm will be used to route requests between all of the local servers. The "weight set" will be refreshed when all local weights reach zero.

### *Affinity*

It is possible to establish an affinity between a client and an application server such that all client requests are sent to the same application server for the life of the affinity. Examples:

- ▶ Transactional (strong) affinity: The first request after the transaction begins will determine to which application server the requests will be routed. All subsequent requests from this client will be routed to the selected application server until the end of the transaction.
- ▶ Applied (weak) affinity: The first time a cluster unaware client needs to use an object, the ORB (via an indirect IOR) will route the request to the LSD. The LSD will return the application server (via a direct IOR) to route the request to. All subsequent requests from this client for that object will be routed to the selected application server.

Thus, the standard weighted round robin algorithm can be overridden by various affinities. In affinity situations, the server weight value is still decremented for each request (starting with WebSphere V5.1), as discussed in 5.4.4, “Plug-in workload management and failover policies” on page 156. However, requests with affinity continue to be sent to the associated application server even when the weight value reaches zero.

#### 9.4.4 LSD failover

If the failure occurs after the routing table is available on the client, the WLMClient code does not go to the LSD to determine what server to route WLM'able objects. The WLMClient code handles the routing decisions. If the failure occurs before the first client request that retrieves the WLM information, then WLM depends on the LSD request to fail over to another LSD.

The WLM component is dependent on LSD failover because the first request for a client will require going to the LSD to obtain the first direct IOR. In order to support clients that do not have any WLM code running on them, for example z/OS clients, other ORB providers, and C++ CORBA clients, the WLM routing decisions will utilize the LSD.

Performance of clients that run in WebSphere Java ORBs with the WLM client code will not be affected at all. The IORs are manipulated and routed by the WLM client code, so indirect and direct IORs are handled the same.

Clients without the WLM code would only route to the LSD on the first request of each object. The direct IOR will be used by the client until the client either drops the reference to the object or the server becomes unavailable.

#### 9.4.5 EJB container failover behavior and tuning

If the failure occurs on the first initial request where the routing table information is not yet available, a COMM\_FAILURE exception will be returned and the ORB will recognize that it has an indirect IOR available and re-send the request to the LSD to determine another server to route to. If the failure occurs after the client retrieved the routing table information, the WLM client will handle the COMM\_FAILURE. The server will be removed from the list of selectable servers and the routing algorithm will be used to select a different server to route the request to.

Consider the following sequence of a client making a request to an application server in the EJB container:

1. For the initial client request, no server cluster and routing information is available in the WLM client's runtime process. The request is therefore directed to the LSD that is hosted on a Node Agent to obtain routing information. If the LSD connection fails, the request will redirect to an alternative LSD (LSD failover, see 9.4.4, "LSD failover" on page 371). If this was not the first request, the WLM client would already have routing information for WLM-aware clients. However, for WLM-unaware clients, the LSD will always route requests to available servers. For future requests from the client, if there is a mismatch of the WLM client's routing information from

what is on a server's, new routing information (as service context) will be added to the response.

2. After getting the InitialContext, a client does a lookup to the EJB's home object (an indirect IOR to the home object). If a failure occurs at this time, the WLM code will transparently redirect this request to another server in the cluster that is capable of obtaining the Bean's home object.
3. Server(s) become unusable during the life cycle of the request.
  - If the request has strong affinity, there cannot be a failover of the request. The request will fail if the original server becomes unavailable. The client must perform recovery logic and resubmit the request.
  - If the request is to an overloaded server, its unresponsiveness makes it seem as though the server is stopped, which may lead to a timeout. In these circumstances it may be helpful to change the server weight and/or tune the ORB and pool properties such as:
    - com.ibm.CORBA.RequestTimeout
    - com.ibm.CORBA.RequestRetriesCount
    - com.ibm.CORBA.RequestRetriesDelay
    - com.ibm.CORBA.LocateRequestTimeout

These can be command-line properties or changed using the Administrative Console.

- If the com.ibm.ejs.wlm.MaxCommFailures threshold has been reached for a cluster member, it is marked unusable. By default, the MaxCommFailures threshold is 0, so that after the first failure the application server is marked unusable. This property can be modified by specifying `-Dcom.ibm.ejs.wlm.MaxCommFailures=<number>` as a command-line argument when launching a client.
- If a machine becomes unreachable (network and/or individual machine errors) before a connection to a server has been established, the operating system TCP/IP keep-alive timeout dominates the behavior of the system's response to a request. This is because a client will wait for the OS-specific, keep-alive, timeout before a failure is detected. This value can be modified, but as described in 5.7.3, "Tuning failover" on page 207, only with caution.

If a connection is already established to a server, com.ibm.CORBA.requestTimeout dominates (the default value is 180 seconds), and a client will wait this length of time before a failure is detected. The default value should only be modified if an application is experiencing timeouts repeatedly, and great care must be taken to tune it properly. If the value is set too high, the failover will be very slow, and set too low, requests will time out before the server has a chance to respond.

The two most critical factors affecting the choice of a timeout value are the amount of time to process a request and the network latency between the client and server. The time to process a request in turn depends on the application and the load on the server. The network latency depends on the location of the client. For example, those running within the same LAN as a server may use a smaller timeout value to provide faster failover. If the client is a process inside of a WebSphere Application Server (the client is a servlet), this property can be modified by editing the request timeout field on the Object Request Broker property sheet. If the client is a Java client, the property can be specified as a runtime option on the Java command line, for example:

```
java -Dcom.ibm.CORBA.requestTimeout=<seconds> MyClient
```

- A failed server is marked unusable, and a JMX notification is sent. The routing table is updated. WLM-aware clients are updated during request/response flows. Future requests will not route requests to this cluster member until new cluster information is received (for example, after the server process is restarted), or until the expiration of the `com.ibm.ejs.wlm.unusable.interval`. This property is set in seconds. The default value is 300 seconds. This property can be set by specifying `-Dcom.ibm.ejs.wlm.unusable.interval=<seconds>` on the command-line arguments for the client process.
- When a request results in an `org.omg.CORBA.COMM_FAILURE` or `org.omg.CORBA.NO_RESPONSE`, the return value of `COMPLETION_STATUS` determines whether a request can be transparently redirected to another server. In the case of `COMPLETED_NO`, the request can be rerouted. If the completed status is `COMPLETED_YES`, no failover is required.

The request was successful, but some communication error was encountered during the marshaling of the response. In the case of `COMPLETED_MAYBE`, WLM cannot verify whether the request was completed successfully, and cannot redirect the request. For example, consider a transaction that must be "at most once". In that case had WLM redirected the request, and it is possible the request would be serviced twice. The programming model is for the client to receive this exception and to have logic in place to decide whether or not to retry the request.

- If all servers are unavailable, the request will result in a `org.omg.CORBA.NO_IMPLEMENT`. At this point, either the network is down, or some other error has caused the entire cluster to be unreachable.

Please note that, similar to the situation of the Web container as discussed earlier, the application servers on a node will be forced to stop when the network is down if the loopback is not configured with the alias of a host IP.

## 9.4.6 Fault isolation and data integrity

There are a lot of data and process interactions between and outside EJB containers. It is important that one client's failure in an EJB container should not impact other clients, and the underlying data must be kept in a consistent way. According to data availability and local data currency, WLM and failover are not supported where fault isolation and data integrity can be broken.

## 9.4.7 EJB caching and failover

An entity bean represents persistent data. It is common for a client to make a succession of requests targeted at the same entity bean instance. It is also possible for more than one client to independently access the same entity bean instance concurrently. The state of an entity bean must be kept consistent across multiple client requests.

Within a transaction, the WLM ensures that the client is routed to the same server. Between transactions, the state of the entity bean can be cached. WebSphere V5.0 supports Option A, Option B, and Option C caching.

Entity beans can be workload managed if they are loaded from the database at the start of each transaction. By providing either Option B caching or Option C caching (the default), entity beans can participate in WLM. These two caching options ensure that the entity bean is always reloaded from the database at the start of each transaction. See 6.2.3, "Entity beans" on page 218 for a detailed description of the three caching options.

Please note that WebSphere V5 also supports the optimistic concurrent control, where the cached data is checked and a collision is detected during the commit stage. Loading data from the database may not be required at the transaction start if the application design is in place to stamp cached entity beans.

## 9.4.8 EJB types and failover

When a client accesses a workload managed EJB, it may be routed to one of a number of instances of the EJB on a number of servers. Not all EJB references can be utilized this way. Table 9-1 shows the types of EJBs that can be workload managed.

*Table 9-1 EJB types and failover*

EJB Type and Option	Type	WLMable
Entity Bean, Option A	Home	Yes
	CMP	No

EJB Type and Option	Type	WLMable
	BMP	No
Entity Bean Options B and C	Home	Yes
	CMP	Yes
	BMP	Yes
Message-Driven Bean	MDB	Yes
Session Bean	Home	Yes
	Stateless	Yes
	Stateful	No

The only type of EJBs that cannot be workload managed are instances of a given stateful session bean and entity beans with commit time caching option A. A stateful session bean is used to capture state information that must be shared across multiple client requests that are part of a logical sequence of operations. To ensure consistency in the logic flow of the application, a client must always return to the same instance of the stateful session bean, rather than having multiple requests workload managed across a group of available stateful session beans.

Because of this restriction, stateful session bean instances are not considered failover tolerant. If the process running the stateful session bean fails, the state information maintained in that bean is unrecoverable. It is the responsibility of the application to reconstruct a stateful EJB with the required state. If this is not the case, an alternative implementation, such as storing state in an Entity EJB, should be considered.

### 9.4.9 Conditions of WLM failover

Automatic failover cannot always happen due to constraints of transaction affinity, server Q-stop, completion status, and server conditions. For situations where WLM cannot determine whether the request is completed, WLM will not automatically fail over the request. It is the client's responsibility to check if a retry is needed.

For example, when an `org.omg.CORBA.COMM_FAILURE` or `org.omg.CORBA.NO_RESPONSE` exception is thrown, the return value of `COMPLETE_STATUS` determines whether automatic failover takes place:

- `COMPLETED_NO`: automatic failover

- ▶ COMPLETED\_YES: no failover
- ▶ MAYBE: no failover

In this case, WLM cannot verify whether the request is completed, and the client will receive a failure and decide whether to retry the request. Also, the client can check the minor codes to determine the status.

#### **9.4.10 Resource redundancy (EJB database, JMS resource, LDAP)**

EJB failover relies on data and resource availability. If the data and resources are unavailable, the EJB client request will fail. For example, if the queue manager fails and the queue has no message(s) for MDB to process, the client will fail. If the database fails, the entity bean cannot complete the transaction, and the EJB client will fail. We discuss high availability of all these in the following sections.

### **9.5 Enhancing WebSphere HA using clustering software**

As discussed in the previous sections, IBM WebSphere Application Server Network Deployment V5.1 has a built-in workload management mechanism that provides excellent scalability and availability. This section discusses combining WebSphere's WLM functionality with clustering software, such as HACMP, VERITAS Cluster Server, Sun Cluster Server, MC/ServiceGuard, and MSCS to further enhance the high availability of WebSphere Application Servers. This two-pronged approach helps resolve the following issues:

- ▶ Two-phase transaction and transaction log failover.
- ▶ Automatic fault detection by employing private heartbeat, hot server standby, and automatic server resource failover.



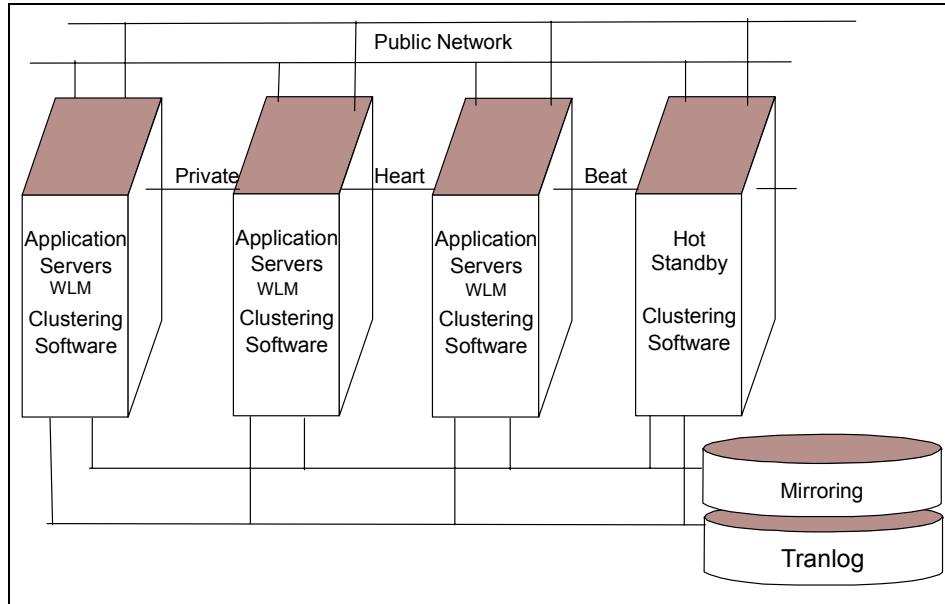


Figure 9-4 IBM WebSphere Application Servers clustered with other clustering software

## 9.5.1 Failover unit

There is various clustering software available to cluster WebSphere Application Servers and clusters. The way clustering solutions are implemented requires what we call a *failover unit* (there is no industry-acceptable term for this concept—each vendor uses a different name, such as package, resource group, service group, or virtual host). This failover unit is treated as a collection of computing resources that needs to run someplace. The operating system (and hardware) clustering solutions take responsibility for getting this collection to run on a system.

From a WebSphere perspective, the important concept is that the WebSphere processes run in a failover unit; they do not run on a machine. The actual machine where the WebSphere processes are running may vary over time, but the failover unit in which they run will not.

In general, a failover unit contains three things:

- ▶ Applications
- ▶ Access points
- ▶ Dependencies

The list of applications for a failover unit contains the WebSphere processes (WebSphere Application Server process, WebSphere deployment management process, WebSphere node agent process, and WebSphere JMS server process). The access points to the applications are the entry points to the WebSphere processes. The dependencies are those things that are unique to a specific WebSphere process and must be moved when the process is moved. An example of such a dependency is the transaction log.

## 9.5.2 Configuration and setup for HACMP

We want to emphasize here that the configuration files of all WebSphere Application Servers must not contain any physical dependent properties such as a physical host IP. You must install and run WebSphere Application Servers using name based virtual hosts (failover unit) in order for the clustering software to work for WebSphere Application Servers independent of physical hosts. In other words, you must change all physical hosts into virtual hosts in your servers and clients or change your host name to virtual. If you want to install more than one unit into a physical host, you need to specify different ports (see the WebSphere InfoCenter for port settings).

Follow these steps to configure and install all necessary software:

1. Install HACMP 4.5 or HACMP 4.3.1 or HACMP 4.4 or HACMP 4.4.1 or HACMP/ES 4.5:
  - Before you configure HACMP, network adapters must be defined, the AIX operating system must be updated, and you must grant clustering nodes permission to access one another. Modify the following configuration files: `/etc/netsvc.conf`, `/etc/hosts`, and `/.rhosts`.  
  
Make sure that each node's service adapters and boot addresses are listed in the `/.rhosts` file on each cluster node so that the `/usr/sbin/cluster/utilities/clruncmd` command and the `/usr/sbin/cluster/godm` command can run.
  - Use `smitt` to install HACMP 4.5, HACMP 4.3.1 or HACMP 4.4 or HACMP 4.4.1 or HACMP/ES 4.5 into both nodes. For installation details, see the *HACMP for AIX Installation Guide* at:  
  
[http://www-1.ibm.com/servers/eserver/pseries/library/hacmp\\_docs.html](http://www-1.ibm.com/servers/eserver/pseries/library/hacmp_docs.html)  
  
You can also install HACMP after you configure the network adapter and shared disk subsystem.
2. Public network configuration:  
  
The public network is used to service client requests (WebSphere Deployment Manager management, WebSphere Application Server, applications, LDAP requests, for example). We defined two TCP/IP public

networks in our configuration. The public network consists of a service/boot adapter and any standby adapters. It is recommended that you use one or more standby adapters. Define standby IP addresses and boot IP addresses. For each adapter, use **smit mktcpip** to define the IP label, IP address, and network mask. HACMP will define the service IP address. Since this configuration process also changes the host name, configure the adapter with the desired default host name last. Then use **smit chinnet** to change service adapters so that they will boot from the boot IP addresses. Check your configuration using **lsdev -Cc if**. Finally, try to ping the nodes to test the public TCP/IP connections.

### 3. Private network configuration:

A private network is used by the cluster manager for the heartbeat message; and can be implemented as a serial network in an HACMP cluster. A serial network allows the cluster manager to continuously exchange keep-alive packets, should the TCP/IP-based subsystem, networks, or network adapters fail. The private network can be either a raw RS232 serial line, a target mode SCSI, or a target mode SSA loop. We used an HACMP for AIX serial line (a null-model, serial to serial cable) to connect the nodes. Use **smit ty** to create the TTY device. After creating the TTY device on both nodes, test the communication over the serial line by entering the command **stty </dev/ttyx** on both nodes (where **/dev/ttyx** is the newly added TTY device). Both nodes should display their TTY settings and return to prompt if the serial line is okay. After testing, define the RS232 serial line to HACMP for AIX.

### 4. Shared disk array installation and LVG configuration:

- The administrative data, application data, session data, LDAP data, transaction log files, other log files and other file systems that need to be highly available are stored in the shared disks that use RAID technologies or are mirrored to protect data. The shared disk array must be connected to both nodes with at least two paths to eliminate the single point of failure. We used IBM 7133 Serial Storage Architecture (SSA) Disk Subsystem.
- You can either configure the shared volume group to be concurrent access or non-concurrent access. A non-concurrent access environment typically uses journaled file systems to manage data, while concurrent access environments use raw logical volumes. There is a graphical interface called TaskGuide to simplify the task of creating a shared volume group within an HACMP cluster configuration. In Version 4.4, the TaskGuide has been enhanced to automatically create a JFS log and display the physical location of available disks. After one node has been configured, import volume groups to the other node by using **smit importvg**.

### 5. Installation of: WebSphere Application Server, WebSphere Network Deployment (or WebSphere Enterprise), DB2 client or Oracle client if using OCI, configuration, instance and database or remote catalog creation:

- For installation details, see the manuals for these products, the WebSphere InfoCenter; or the redbook *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195. You can install the products either on local disks or on the shared disk array.
- As mentioned before, you need to use virtual host names for this setup. It is better to change to using virtual host names *before* you install WebSphere. Otherwise you may need to manually change the virtual host names in all WebSphere configuration XML files afterwards. (Doing so is not recommended nor is it supported at present, though a set of scripts for doing so are slated for Web delivery as part of WebSphere Application Server V5.1.1.)

**Important:** Although you can install the WebSphere code on each node's local disks, it is very important to keep all *shared data*, such as the configuration repository and log files, on the shared disk array. This makes sure that another node can access this data when the current Deployment Manager node fails.

You will need to install the binaries for WebSphere Application Server twice:

- Mount the shared disk array to the first node, for example as /ha. Install WebSphere Application Server into /ha as root, so the installation path is /ha/WebSphere/AppServer.
- After installation, start server1 to make sure everything is OK.
- Delete all /ha/WebSphere/AppServer files.
- Mount the shared disk array to the same mount point on the other node, which again is /ha. Install WebSphere into /ha on the second node.
- When the installation is completed on the second node, start server1 and test the environment again.

Doing this makes sure that all nodes pick up the WebSphere packages correctly.

**Note:** If you choose to install the software onto local disks rather than on the shared disk array, then you must install the same software version on both systems!

- Install the WebSphere Network Deployment code on a separate system or on one of your WebSphere Application Server nodes.

**Note:** You might also want to make the Deployment Manager highly available. Instructions for doing this are found in 10.5.2, “Make Deployment Manager highly available using clustering software and hardware” on page 406.

- Use **addNode.sh** to federate your WebSphere Application Server nodes to your Deployment Manager. The Node Agent is created and started automatically during the federation operation.
  - Create WebSphere clusters and cluster members.
  - Be aware that you cannot fail over individual parts of the environment, for example an application server. You always have to failover the entire WebSphere Failover Unit. For example:
    - The WebSphere Application Servers must failover together with their Node Agent.
    - All application servers in a node must failover together as a WebSphere Failover Unit.
    - Any application server must failover together with all of its dependent resources.
  - If “thick” database clients are used, install the database clients on your WebSphere nodes and configure the database clients to connect to the database server. For example, install DB2 clients on all WebSphere nodes and catalog the remote node and database server.
    - The DB2 client must also failover together with the application servers.
6. Define the cluster topology and HACMP application servers.
- Please note that an HACMP application server is different from a WebSphere Application Server. In our case, the HACMP application server is the WebSphere Failover Unit that includes all the components as mentioned above.
7. Define the cluster topology and HACMP application servers - please note that HACMP application server is totally different from WebSphere Application Server - We call HACMP application server as WebSphere Failover Unit:
- The cluster topology is comprised of cluster definition, cluster nodes, network adapters, and network modules. The cluster topology is defined by entering information about each component in HACMP-specific ODM classes. These tasks can be performed using **smit hacmp**. For details, see the *HACMP for AIX Installation Guide* at:

[http://www-1.ibm.com/servers/eserver/pseries/library/hacmp\\_docs.html](http://www-1.ibm.com/servers/eserver/pseries/library/hacmp_docs.html)

- An “application server”, in HACMP or HACMP/ES, is a cluster resource that is made highly available by the HACMP or HACMP/ES software. For example in this case, WebSphere Application Servers, Node Agents, Deployment Manager, database client. Do not confuse this with WebSphere Application Server. Use **smitty hacmp** to define the HACMP (HACMP/ES) application server with a name and its start script and stop script.
- Start and stop scripts for WebSphere Failover Unit (WFOU) as the HACMP application servers.

Our sample WFOU service start script is:

```
/ha/WebSphere/AppServer/bin/startNode.sh
/ha/WebSphere/AppServer/bin/startServer.sh WLMmember1
/ha/WebSphere/AppServer/bin/startServer.sh WLMmember2
su - db2inst1 <<MYDB2
db2start
MYDB2
```

And our sample WFOU service stop script is:

```
/ha/WebSphere/AppServer/bin/stopNode.sh
/ha/WebSphere/AppServer/bin/stopServer.sh WLMmember1
/ha/WebSphere/AppServer/bin/stopServer.sh WLMmember2
su - db2inst1 <<MYDB2
db2 force applications all
db2stop
MYDB2
```

You must test each configuration part/script individually before you can bundle all of them as the Failover Unit.

Remember that the WebSphere Failover Unit includes the WebSphere Application Server, Node Agent(s), and database client. You must failover all of these together as an integrated failover unit, an individual server failover will not work.

8. Define and configure the resource groups:
  - For HACMP and HACMP/ES to provide a highly available application server service, you need to define the service as a set of cluster-wide resources essential to uninterrupted processing. The resource group can have both hardware and software resources such as disks, volume groups, file systems, network addresses, and application servers themselves.
  - The resource group is configured to have a particular kind of relationship with a set of nodes. There are three kinds of node relationships: cascading, concurrent access, or rotating. For the cascading resource group, setting the cascading without fallback (CWOFF) attribute will minimize the client failure time. We used this configuration in our tests.

Use **smit** to configure resource groups and resources in each group. Finally, you need to synchronize cluster resources to send the information contained on the current node to the other node.

9. Cluster verification:

- Use **/usr/sbin/cluster/daig/clverify** on one node to check that all cluster nodes agree on the cluster configuration and the assignment of HACMP for AIX resources. You can also use **smit hacmp** to verify the cluster. If all nodes do not agree on the cluster topology and you want to define the cluster as it is defined on the local node, you can force agreement of cluster topology onto all nodes by synchronizing the cluster configuration. Once the cluster verification is satisfactory, start the HACMP cluster services using **smit hacmp** on both nodes, and monitor the log file using **tail -f /tmp/hacmp.out**. Check the database processes.

10. Takeover verification:

- To test a failover, use **smit hacmp** to stop the cluster service with the takeover option. On the other node, enter the **tail -f /tmp/hacmp.out** command to watch the takeover activity.

Example 9-10 shows the complete configuration file for a successful WebSphere Application Server failover.

*Example 9-10 Configuration script for WebSphere Application Servers Failover Unit*

---

```
Cluster Description of Cluster hacluster
Cluster ID: 99
There were 2 networks defined : haserial, tr157
There are 2 nodes in this cluster.

NODE hacmp1:
    This node has 2 service interface(s):

        Service Interface hacmp1-tty:
            IP address:      /dev/tty0
            Hardware Address:
            Network:         haserial
            Attribute:       serial
            Aliased Address?: False

        Service Interface hacmp1-tty has no boot interfaces.
        Service Interface hacmp1-tty has no standby interfaces.

Service Interface hacmp1s:
IP address:      10.2.157.68
Hardware Address:
Network:         tr157
```

```

        Attribute:      public
        Aliased Address?:      False
Service Interface hacmp1s has 1 boot interfaces.
        Boot (Alternate Service) Interface 1: hacmp1b
        IP address:      10.2.157.207
        Network:      tr157
        Attribute:      public
        Service Interface hacmp1s has 1 standby interfaces.
        Standby Interface 1: hacmp1sb
        IP address:      10.10.0.30
        Network:      tr157
        Attribute:      public

```

```

NODE hacmp2:
    This node has 2 service interface(s):

```

```

    Service Interface hacmp2-tty:
        IP address:      /dev/tty0
        Hardware Address:
        Network:      haserial
        Attribute:      serial
        Aliased Address?:      False

```

```

    Service Interface hacmp2-tty has no boot interfaces.
    Service Interface hacmp2-tty has no standby interfaces.

```

```

Service Interface hacmp2s:
    IP address:      10.2.157.196
    Hardware Address:
    Network:      tr157
    Attribute:      public
    Aliased Address?:      False
Service Interface hacmp1s has 1 boot interfaces.
    Boot (Alternate Service) Interface 1: hacmp1b
    IP address:      10.2.157.207
    Network:      tr157
    Attribute:      public
Service Interface hacmp1s has 1 standby interfaces.
    Standby Interface 1: hacmp1sb
    IP address:      10.10.0.30
    Network:      tr157
    Attribute:      public

```

```

NODE hacmp2:
    This node has 2 service interface(s):

```



Service Interface hacmp2-tty:  
IP address: /dev/tty0  
Hardware Address:  
Network: haserial  
Attribute: serial  
Aliased Address?: False

Service Interface hacmp2-tty has no boot interfaces.  
Service Interface hacmp2-tty has no standby interfaces.

Service Interface hacmp2s:  
IP address: 10.2.157.196  
Hardware Address:  
Network: tr157  
Attribute: public  
Aliased Address?: False

Service Interface hacmp2-tty has no boot interfaces.  
Service Interface hacmp2-tty has no standby interfaces.

Service Interface hacmp2s:  
IP address: 10.2.157.196  
Hardware Address:  
Network: tr157  
Attribute: public  
Aliased Address?: False

Service Interface hacmp2s has 1 boot interfaces.  
Boot (Alternate Service) Interface 1: hacmp2b  
IP address: 10.2.157.206  
Network: tr157  
Attribute: public  
Service Interface hacmp2s has 1 standby interfaces.  
Standby Interface 1: hacmp2sb  
IP address: 10.10.0.40  
Network: tr157  
Attribute: public

Breakdown of network connections:

Connections to network haserial

Node hacmp1 is connected to network haserial by these interfaces:  
hacmp1-tty

Node hacmp2 is connected to network haserial by these interfaces:  
hacmp2-tty

Connections to network tr157

Node hacmp1 is connected to network tr157 by these interfaces:

hacmp1b  
hacmp1s  
hacmp1sb

Node hacmp2 is connected to network tr157 by these interfaces:

hacmp2b  
hacmp2s

hacmp2sb

Resource Group Name	WASND
Node Relationship	cascading
Participating Node Name(s)	hacmp1 hacmp2
Dynamic Node Priority	
Service IP Label	hacmp1s
Filesystems	/ha
Filesystems Consistency Check	fsck
Filesystems Recovery Method	sequential
Filesystems/Directories to be exported	
Filesystems to be NFS mounted	
Network For NFS Mount	
Volume Groups	havg
Concurrent Volume Groups	
Disks	
Connections Services	
Fast Connect Services	
Shared Tape Resources	
Application Servers	WASFOU
Highly Available Communication Links	
Miscellaneous Data	
Auto Discover/Import of Volume Groups	true
Inactive Takeover	false
Cascading Without Fallback	true
9333 Disk Fencing	false
SSA Disk Fencing	false
Filesystems mounted before IP configured	true
Run Time Parameters:	
Node Name	hacmp1
Debug Level	high
Host uses NIS or Name Server	false
Format for hacmp.out	Standard
Node Name	hacmp2
Debug Level	high

Host uses NIS or Name Server	false
Format for hacmp.out	Standard

---

### 9.5.3 Failover process

If a network disk fails or applications, logs, or data cannot be retrieved locally, a process can be available automatically on the hot standby node, where these resources have been replicated and are immediately available. The maximum loss of any given resource depends on the frequency of replication of information. Further, if a node fails due to CPU load, an operating system, or power failure, or due to a JVM failure, the process can be available on a hot standby node. Finally, if the network fails, a backup network connection can be made available. If the backup network is also experiencing difficulties, a process can become available on a hot standby node on a different network.

### 9.5.4 Advantages

WebSphere's workload management provides excellent failover and scalability support for almost all applications. However, with the addition of HACMP, there are some advantages:

- ▶ Provides two-phase commit transaction and transaction log automatic failover. WebSphere Application Servers act as coordinators of distributed transactions and write the transaction logs in the `<Install_Root>/WebSphere/AppServer/tranlog` directory. If an application server fails after the start of the two-phase commit, the transaction manager will wait for the application to be restarted to complete the transaction. If the node hosting the application server fails, the transaction cannot be completed, and the corresponding database rows are locked until the transaction can be completed. With IP takeover, a new application server is restarted, with access to the tranlog, and the transaction can be completed automatically.
- ▶ Provides automatic application server failover if a machine has an unrecoverable error such that the client will not perceive a performance impact during failover. Without an HACMP cluster, the machine that encounters an error (and hosting an application server that is servicing requests) needs to be manually rebooted to recover the server process.

Although such configuration offers the advantages mentioned above, it is a highly complex configuration.





## Deployment Manager and Node Agent high availability

In IBM WebSphere Application Server Network Deployment V5.1, the distributed administrative work is accomplished by the Node Agent server that resides on each node and the Deployment Manager that acts as a central point for administrative tasks.

In this chapter, we describe how failures of a Node Agent server or the Deployment Manager server affect the different types of clients, and how to enable high availability.

## 10.1 Introduction

The Node Agent server and the Deployment Manager server both use a repository of XML files on their own nodes. The master repository data is stored on the Deployment Manager node. That data is then replicated to each node in the Administrative domain (or cell). The Deployment Manager server and the Node Agent servers keep these files synchronized. The default synchronization interval is one minute. The synchronization process is unidirectional from the Deployment Manager to the Node Agents to ensure repository integrity. That means that any changes made on the Node Agent level are only temporary and will be overwritten by the Deployment Manager during the next synchronization. Only changes to the master data through the Deployment Manager are permanent and replicated to each node.

Administrative and configuration actions as well as operational changes can be applied to either the Node Agent server, the application server, or the Deployment Manager through wsadmin scripting. To communicate with these servers you can use either the SOAP or the RMI protocol. The Administrative Console (adminconsole.ear) is an application that is installed in the Deployment Manager server by default in a WebSphere Network Deployment environment.

The Node Agent server provides runtime support for the Location Service Daemon (LSD), file transferring and synchronization, JMX server, distributed logging, naming server, security server, and EJB workload management configuration.

The Deployment Manager provides runtime support for WLM services, file transferring and synchronization, configuration management, JMX server, distributed logging, naming server, security server, and for the master configuration.

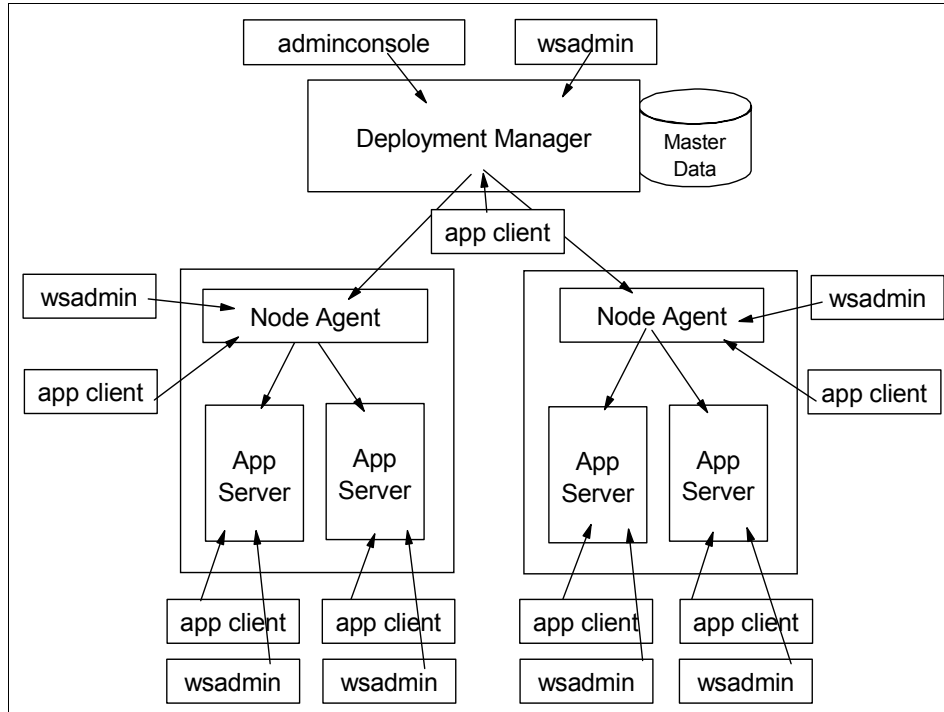


Figure 10-1 WebSphere Deployment Manager and Node Agent for configuration administration and application bootstrapping

## 10.2 Node Agent failures

The Node Agent provides several important services to the Deployment Manager, application servers, and to the application clients. These services include administrative services, the Location Service Daemon, file transfer service, synchronization service, naming service, security service, and RAS service.

There are six types of errors in the process, network, disk, and machine that may cause a Node Agent server to fail:

- ▶ Expected server process failure, for example stopping the server.
- ▶ Unexpected server process failure, for example the server JVM crashes, simulated by kill -9.
- ▶ Server network problem, for example the network cable is disconnected or a router is broken.

- ▶ Unexpected and expected machine problems, for example a machine shutdown, operating system crashes, or power failures.
- ▶ Disk is faulty and the configuration file cannot be read.
- ▶ Overloading with heavy hacking or denial of service (DOS) attacks, simulated by continuously running wsadmin to the Node Agent using Mercury LoadRunner.

We discuss the behaviors of Node Agent server failures and how WebSphere V5 mitigates the impacts of these failures on WebSphere Application Servers and application clients. Also, we describe how to set up Node Agents to tolerate these failures or to minimize the impact of these failures.

## 10.2.1 Application servers

The Node Agent hosts the Location Service Daemon, publishes and synchronizes configuration data to other application servers, and monitors and launches the managed application server process.

- ▶ An application server can only be started if the local Node Agent is available. If the Node Agent is not active, the application server cannot register itself with the LSD and you will receive the error message in Example 10-1.

### *Example 10-1 Error message at application server startup with inactive Node Agent*

---

```
WsServer      E WSVR0003E: Server WebHAbbMember3 failed to start
com.ibm.ws.exception.RuntimeError
    at com.ibm.ws.runtime.component.ORBImpl.start(ORBImpl.java:227)
    at
com.ibm.ws.runtime.component.ContainerImpl.startComponents(ContainerImpl.java:3
43)
    at com.ibm.ws.runtime.component.ContainerImpl.start(ContainerImpl.java:234)
    at com.ibm.ws.runtime.component.ServerImpl.start(ServerImpl.java:180)
    at com.ibm.ws.runtime.WsServer.start(WsServer.java:135)
    at com.ibm.ws.runtime.WsServer.main(WsServer.java:232)
    at java.lang.reflect.Method.invoke(Native Method)
    at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:94)
---- Begin backtrace for nested exception
com.ibm.ejs.EJSEException: Could not register with Location Service Daemon;
nested exception is:
    org.omg.CORBA.TRANSPORT: Host unreachable:
connect:host=10.2.90.84,port=9900  minor code: 4942F301  completed: No
org.omg.CORBA.TRANSPORT: Host unreachable: connect:host=10.2.90.84,port=9900
minor code: 4942F301  completed: No
    at
com.ibm.CORBA.transport.TransportConnectionBase.connect(TransportConnectionBase
.java:338)
```



```

    at
com.ibm.ws.orbimpl.transport.WSTransport.getConnection(WSTransport.java:412)
    at com.ibm.rmi.iiop.TransportManager.get(TransportManager.java:84)
    at com.ibm.rmi.iiop.GIOPImpl.locate(GIOPImpl.java:177)
    at com.ibm.rmi.corba.Corbaloc.locateUsingINS(Corbaloc.java:301)
    at com.ibm.rmi.corba.Corbaloc.resolve(Corbaloc.java:363)
    at
com.ibm.rmi.corba.InitialReferenceClient.resolve_initial_references(InitialReferenceClient.java:151)
    at com.ibm.rmi.corba.ORB.resolve_initial_references(ORB.java:3148)
    at com.ibm.rmi.iiop.ORB.resolve_initial_references(ORB.java:531)
    at com.ibm.CORBA.iiop.ORB.resolve_initial_references(ORB.java:2884)
    at com.ibm.ejs.oa.LocationService.register(LocationService.java:160)
    at com.ibm.ejs.oa.EJSServerORBImpl.<init>(EJSServerORBImpl.java:137)
    at com.ibm.ejs.oa.EJSORB.init(EJSORB.java:334)
    at com.ibm.ws.runtime.component.ORBImpl.start(ORBImpl.java:223)
    at
com.ibm.ws.runtime.component.ContainerImpl.startComponents(ContainerImpl.java:343)
    at
com.ibm.ws.runtime.component.ContainerImpl.start(ContainerImpl.java:234)
    at com.ibm.ws.runtime.component.ServerImpl.start(ServerImpl.java:180)
    at com.ibm.ws.runtime.WsServer.start(WsServer.java:135)
    at com.ibm.ws.runtime.WsServer.main(WsServer.java:232)
    at java.lang.reflect.Method.invoke(Native Method)
    at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:94)

```

WsServer      E WSVR0009E: Error occurred during startup

---

- ▶ Once the application server is started, it will run normally even if the Node Agent becomes unavailable. However, configuration data cannot be updated consistently.
- ▶ The application server processes are managed by the Node Agent server. When the Node Agent is unavailable, the application server loses the benefit of being a managed server. Therefore, an application cannot be restarted automatically if the application server dies unintentionally. You can tune the managed-by-nodeagent server process monitoring parameters on the Administrative Console as shown in Figure 10-2 on page 394.

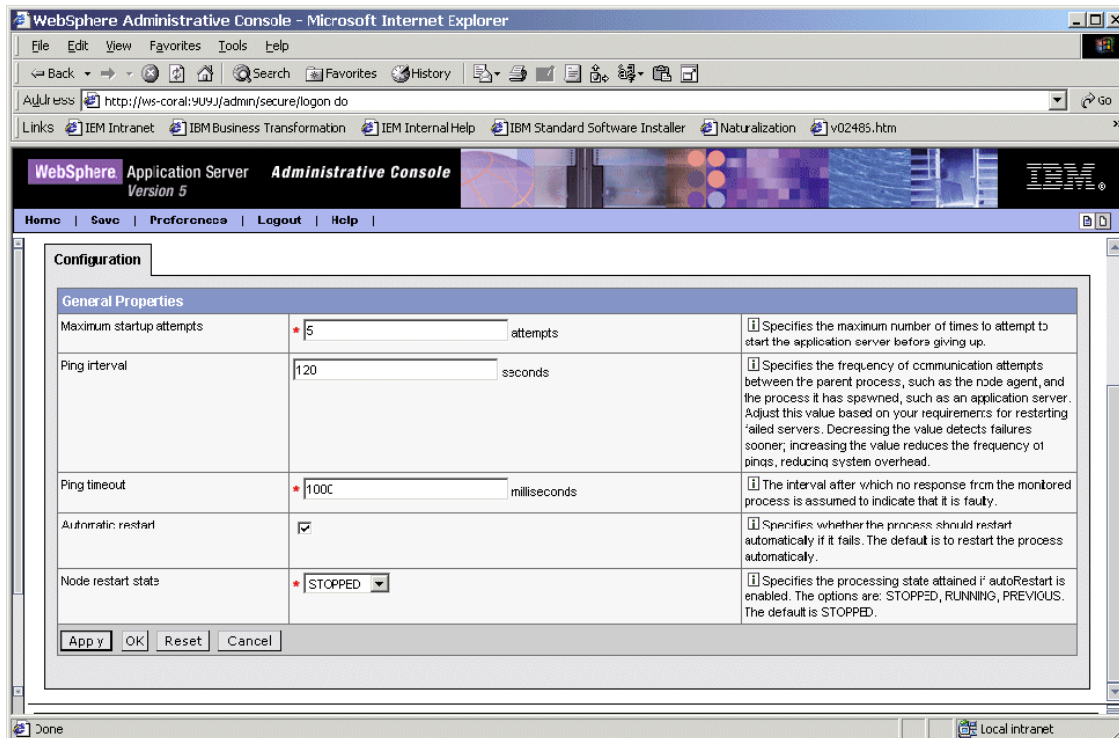


Figure 10-2 Managed server monitoring parameters tuning

It is recommended that each application server on a node be restarted if the Node Agent is restarted.

## 10.2.2 Deployment Manager

Node Agents and the Deployment Manager work together to manage administrative tasks and to make configuration and operational changes.

If a Node Agent server is not available, an error message similar to the following is shown:

```
RoutingTable W ADMCO047W: Lost connection to the node agent at node
"ws-squirrel". The routing table is updated to remove the entry for this
node.
RoutingTable W ADMCO047W: Lost connection to the node agent at node
"ws-volcano". The routing table is updated to remove the entry for this
node.
```

If a Node Agent fails, the Deployment Manager may remove servers on that node from its WLM routing table and will publish this information to other nodes where the Node Agent is still active.

Administrative tasks from the Deployment Manager to that node will fail if the Node Agent is not available, any configuration and operational changes will not be reflected in this node. Once the failed Node Agent recovers, the routing table will be updated and published to all other nodes again.

```
RoutingTable A ADMC0048I: Established the connection to previously  
disconnected nodeagent at node ws-squirrel  
RoutingTable A ADMC0048I: Established the connection to previously  
disconnected nodeagent at node ws-volcano
```

### 10.2.3 Location Service Daemon

The LSD was redesigned in WebSphere V5.1 to mitigate the impact of Node Agent failures. We rely on multiple Node Agents and built-in HA LSD to provide high availability. The Node Agent itself is not workload managed, but WebSphere V5.0 provides the mechanism to make indirect IORs (Interoperable Object Reference) aware of all LSDs.

### 10.2.4 Naming server

In IBM WebSphere Application Server Network Deployment V5.1, a naming server exists in every server process (application server, Node Agent, and Deployment Manager). You can bootstrap your client to any of these naming servers. Usually servlets, EJB clients, and J2EE clients start their bootstrap from their local application server and end up on the server where the objects are found.

For Java clients, you can bootstrap to more than one naming server on different Node Agents as shown in Example 10-2.

*Example 10-2 Java client bootstrapping*

---

```
prop.put(Context.INITIAL_CONTEXT_FACTORY,  
"com.ibm.websphere.naming.WsnInitialContextFactory");  
prop.put(Context.PROVIDER_URL, "corbaloc::host1:2809,host2:2809");  
Context initialContext = new InitialContext(prop);  
try { java.lang.Object myHome =  
initialContext.lookup("cell/clusters/MyCluster/MyEJB");  
myHome = (myEJBHome) javax.rmi.PortableRemoteObject.narrow(myHome,  
myEJBHome.class);  
} catch (NamingException e) { }
```

---

The client automatically tries the bootstrap servers on host1 and host2 until it gets an InitialContext object. The order in which the bootstrap hosts are tried is not guaranteed.

The Node Agent is not workload managed. If a client gets a cached InitialContext object and that Node Agent is unavailable, the InitialContext object will not automatically fail over to another Node Agent, so the lookup with this InitialContext object will fail. In order to avoid this problem, you need to disable the cache in your client by supplying the following property when initializing the naming context:

```
prop.put (PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_NONE)
```

Where PROPS.JNDI\_CACHE\_OBJECT is a Java constant defined in com.ibm.websphere.naming.PROPS.

Or more conveniently by setting the Java command line property as:

```
java -Dcom.ibm.websphere.naming.jndicacheobject=none MyClient
```

Using Node Agents as bootstrap servers is not recommended because they are not workload managed. Because application servers in a cluster are workload managed, you should use application servers as bootstrap servers instead as follows:

```
prop.put(Context.PROVIDER_URL, "corbaloc::host1:9810,:host1:9811,  
:host2:9810, :host2:9811");
```

The above discussion is only relevant for naming lookup (read) operations. If your application is binding (writing), a failure will occur if the Node Agent is unavailable, since the Node Agent coordinates the binding process.

## 10.2.5 Security server

Security servers are also present in every application server process in WebSphere V5. Security credentials can fail over from one server to another in a cluster until they are expired. However, the LDAP server is a SPOF. Refer to Chapter 13, “High availability of LDAP, NFS, and firewall” on page 503 for information about how to enable LDAP high availability.

## 10.2.6 Application clients

An application client receives its routing information from the LSD (routing table) hosted in the Node Agent. If the application server environment does not have routing information for a client failover available, the client will fail to run if all Node Agents are unavailable. Once the application server environment has routing information available, the application client will run successfully if it is not

using a single Node Agent to bootstrap and is not binding (writing). All EJB IORs contain the list of LSD host and ports.

However, the Deployment Manager may remove servers on a node with a failed Node Agent and publish a new routing table to all nodes where Node Agents are still alive, which disturbs the WLM routing.

### **10.2.7 Synchronization Service and File Transfer Service**

All the following services will fail to function when the Node Agent is unavailable.

### **10.2.8 RAS service, PMI and monitoring**

A node with a failed Node Agent will not be able to provide RAS service, PMI, and monitoring information to the cell.

### **10.2.9 Administrative clients**

There are two kinds of administrative clients: the Administrative Console and the wsadmin scripting interface. wsadmin can be attached to any server (Node Agent, Deployment Manager, or application servers). The Administrative Console is a Web application that is usually installed in the Deployment Manager server in a WebSphere Network Deployment environment.

#### **Administrative Console**

When the Node Agent is unavailable, you will see the error message "lost contact with node agent xx" in the Deployment Managers SystemOut.log. The following limitations when working with the Administrative Console apply when a Node Agent is not available:

- ▶ Any query for any information on that node will fail and leave the server status for that node as "unknown" or "unavailable".
- ▶ Any configuration changes will not appear on that node until the Node Agent is restarted.
- ▶ Any operational actions (start, stop, delete, create) from the Administrative Console to the servers on the node with a failed Node Agent will fail.

However, you can do any administrative and configuration tasks using the Administrative Console for other nodes where the Node Agents are still alive.

#### **wsadmin scripting interface**

You can connect the wsadmin scripting interface to every server in the WebSphere administrative domain (or cell). By default, wsadmin is attached to

the Deployment Manager. You can change the default by editing the wsadmin properties file located in the <install\_root>/properties directory or you can specify the conntype, host, and port parameters in the command line:

```
wsadmin -conntype RMI -host mynode -port 2809
```

You can look up the conntype and port in the SystemOut.log. For example:

```
JMXSoapAdapte A ADMC0013I: SOAP connector available at port 8878
RMICConnectorC A ADMC0026I: RMI Connector available at port 2809
```

Obviously, wsadmin cannot connect to the Node Agent when it is unavailable.

## 10.3 Enhancing Node Agent high availability

In WebSphere Network Deployment, we usually have more than one Node Agent in the cell, and HA LSD and IOR contain the information of all nodes, so another Node Agent can service client requests if one Node Agent fails. Therefore, having more than one Node Agent in a cell is the basic solution to ensure that LSD failover will occur if a Node Agent process is lost.

The Node Agent also monitors its managed application servers and maintains configuration data currency and integrity in runtime. If the Node Agent is unavailable, the application servers will lose the protection and the configuration data may become stale.

### Loopback alias configuration

Expecting an application server to run when the machine is disconnected from the network requires that the loopback port address (127.0.0.1) is configured as an alias endpoint for the machine. The reason for this is that the Node Agent normally pings all application servers on the node. When the Node Agent detects that an application server is not available, it stops and tries to restart the application server. This goes on for 10 minutes, at which time the Node Agent no longer tries to restart the application server. The ping from the Node Agent to the application server uses the Internet Protocol (IP). If the machine is disconnected from the network, the communication occurs on the loopback port. However, the loopback port is only considered a valid port for the application server when the loopback port address (127.0.0.1) is configured as an alias for the machine.

The behavior when the loopback port is not configured as an alias is that the application server is stopped (because the Node Agent cannot reach the application server), and the Node Agent will not be able to restart it until the machine is again connected to the network. If the time required to reconnect to the network is less than 10 minutes, then the application servers will be started on the next retry of the Node Agent. However, when the reconnection time

exceeds the 10-minute interval, then an explicit start of the application servers has to be performed.

### 10.3.1 Add Node Agent as OS daemon

When the Node Agent process crashes and stops unintentionally, the operating system service will restart the Node Agent server automatically.

For the Windows platform, you can add the Node Agent as an operating system daemon using:

```
WASService -add <Nodeagent> -serverName Nodeagent -restart true
```

For AIX, you can do so by adding an entry to the inittab file:

```
was:2:once:/usr/WebSphere/AppServer/bin/rc.was >dev/console 2>&1
```

However, if the disk where the configuration files reside is defective, or the JVM crashes, or the operating system itself crashes, or the machine dies, the Node Agent will not recover until the manual recovery process has finished.

### 10.3.2 Enhancing Node Agent HA using clustering software

We can use HACMP, VERITAS Cluster Server, Sun Cluster Server, MC/ServiceGuard, or Microsoft Cluster Service to enhance Node Agent high availability. The configuration is similar to the configuration options discussed in Chapter 9, “WebSphere Application Server failover and recovery” on page 345 and Chapter 12, “WebSphere data management high availability” on page 435, where you find step-by-step setup and configuration instructions.

The typical Node Agent clustering configuration is illustrated in Figure 10-3 on page 400.

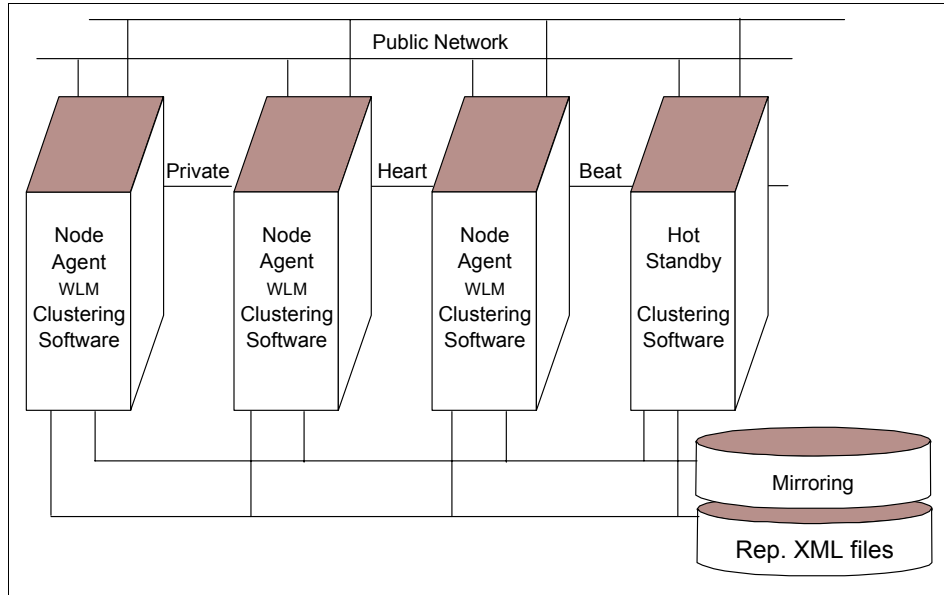


Figure 10-3 Node Agent clustering with cluster software

Usually, you will include the Node Agents in the same failover unit as the application servers, with the same virtual IP address access point. You store its configuration files and other dependencies in a shared disk array.

## 10.4 Deployment Manager failures

The Deployment Manager is the central control point for all administrative and configurational tasks as well as operational actions. It provides a single image of the whole cell and a mechanism to administer the entire cell.

The Deployment Manager provides several important services to application servers, Node Agents, and application clients. It supports WLM services, administrative services, file transfer service, synchronization service, naming service, security service, PMI, and RAS service.

There are six types of failures in process, network, disk, and machine that may cause the Deployment Manager server to fail:

- ▶ Expected server process failure, for example stopping the server.
- ▶ Unexpected server process failure, for example the server JVM crashes, simulated by kill -9.



- ▶ Server network problem, for example a network cable is disconnected or a router is broken.
- ▶ Unexpected and expected machine problem, for example a machine shutdown, operating system crashes, or power failures.
- ▶ Disk is defective and the configuration file cannot be read.
- ▶ Overloading with heavy hacking or denial of service (DOS) attacks, simulated by continuously running wsadmin to the Node Agent using LoadRunner.

The Deployment Manager is not clustered and therefore is a SPOF in WebSphere V5.1. However, its impact on the application client processing is limited because all configuration data is replicated to every Node Agent in the cell (although the configuration data may be stale or not current).

As soon as the Deployment Manager server becomes available again, all Node Agents in the domain will discover it, and all functions will return to normal.

In the following sections, we discuss the behaviors of Deployment Manager server failures, how WebSphere V5.0 mitigates the impacts of these failures on application servers and application clients, and how to set up the Deployment Manager to tolerate these failures or to minimize the impact of these failures.

## 10.4.1 Configuration management

In WebSphere V4, a common repository database was used to link all nodes in an administrative domain. There were no repository synchronization and data integrity problems because the database handles this. In WebSphere V5.0, each individual node is "added" to the Deployment Manager cell. All nodes in the domain are federated to create a master repository of XML files on the Deployment Manager node. The master repository is then replicated to all other nodes in the domain. The Deployment Manager keeps all copies of the configuration files synchronized across all nodes in the domain. During normal operation, the Deployment Manager consistently synchronizes the repository data on each node with the master repository. In addition to the configuration repository, you have the option to also synchronize application binaries.

You can turn synchronization on/off and change configuration parameters to tune synchronization using the Administrative Console.

WebSphere V5 also provides the option to manually synchronize copies of the repository and binaries by using the command:

```
<install_root>\WebSphere\AppServer\bin\syncNode.bat
```

or by using the Administrative Console as shown in Figure 10-4 on page 402.

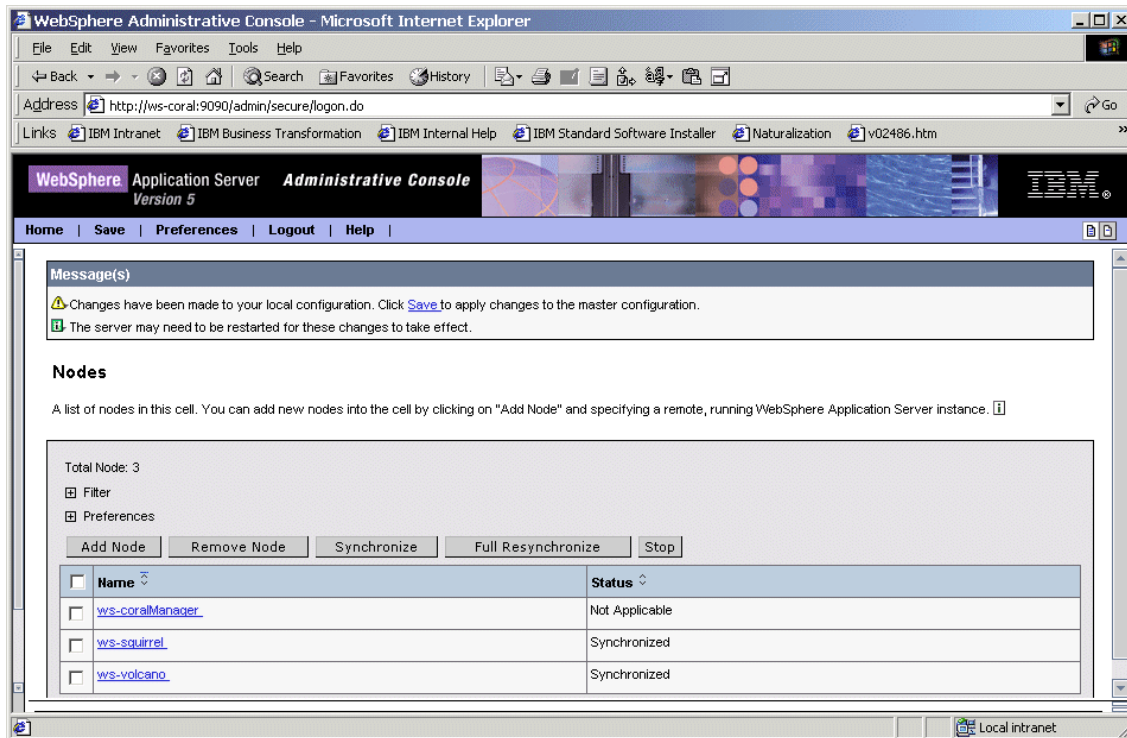


Figure 10-4 Node synchronization

**Note:** If the Deployment Manager is not available, the repository cannot be synchronized. You can use wsadmin to change the configuration at the node level. Repository files across all nodes in the domain may become inconsistent or stale. Once the Deployment Manager comes online again, all changes made to Node Agents or application servers are lost, and are overwritten with the master repository.

## 10.4.2 Node Agent

The Node Agent interacts directly with the Deployment Manager to perform all administrative and configuration tasks. If the Deployment Manager is unavailable, all cell-wide tasks cannot be executed and all tasks done locally will be overwritten by the master repository in the Deployment Manager node.

When the Deployment Manager is not available, configuration synchronization will fail with the following error message:

```
NodeSync      E ADMS0015E: The synchronization request can not be completed
               because the node agent can not communicate with the deployment manager.
NodeSyncTask  A ADMS0016I: Configuration synchronization failed.
```

The Deployment Manager and the Node Agents can be started in any order. They will discover each other as soon as they are started.

### 10.4.3 Application server

Application servers have no direct interaction with the Deployment Manager. However, they rely on up-to-date configuration data and on the fact that the Deployment Manager server replicates and initiates any changes. For example, the Deployment Manager controls the WLM master routing table. Application servers will not be informed of failed servers on other nodes if the Deployment Manager fails, and client requests may be routed to failed servers.

### 10.4.4 Naming server

The Deployment Manager also hosts a cell naming server that controls the whole naming context structure.

Clients can bootstrap to the cell naming server using:

```
prop.put(Context.PROVIDER_URL, "corbaloc::dmgrhost:9809");
```

The Deployment Manager server is not workload managed in WebSphere V5.0, and neither is the naming service inside the Deployment Manager process. Unless you have a highly available Deployment Manager with HACMP, MSCS, VERITAS Cluster, ServiceGuard, or Sun Cluster, you should not use it to bootstrap your application clients. However, a highly available Deployment Manager has the advantage of a single image of the domain.

No matter where your client bootstraps, any naming binding or update writing at cell scope will fail if the Deployment Manager is not available.

### 10.4.5 Security server

The cell security server is not available when the Deployment Manager is down. This has no impact on application clients since they use the local security service in every application server. Nevertheless, the LDAP server does present a SPOF for security servers. See Chapter 13, “High availability of LDAP, NFS, and firewall” on page 503 for more information about high availability of LDAP servers.

## 10.4.6 WLM runtime service

When the Deployment Manager is not available, the cell WLM runtime service that maintains and publishes the master routing table is also not available. Any cluster server operational status changes (such as start/stop servers) cannot be reflected on the master routing table, and cluster servers on different nodes will not get configuration updates.

## 10.4.7 Application clients

There is very limited direct impact on application clients when the Deployment Manager is not available. They will notice the failure only if they are bootstrapped to the cell naming server or when the application clients do bindings and updates. However, application clients may be impacted indirectly by unavailable configuration and operational services due to the failed Deployment Manager, for example inconsistent configuration data, stale WLM routing table, or aged application binaries.

## 10.4.8 Synchronization Service and File Transfer Service

The Deployment Manager provides file transfer and synchronization of configuration files and application binaries across nodes in the domain. These services are not available when the Deployment Manager is down.

## 10.4.9 RAS Service and PMI monitoring

The cell-wide information and remote logs are not available when the Deployment Manager is unavailable.

## 10.4.10 Administrative clients

This section discusses the impacts of Deployment Manager failures on the Administrative Console and the wsadmin scripting interface.

### **Administrative Console**

The default adminconsole application cannot be started if the Deployment Manager is unavailable.

### **wsadmin scripting interface**

By default, you are connected to the Deployment Manager server. If it is unavailable, wsadmin will fail. You can change the wsadmin properties file or add parameters on the command line to attach wsadmin to a Node Agent server or application server.

You can look up the wsadmin connect types and ports in the SystemOut.log file.

► For an application server:

SOAP connector available at port 8881

RMI Connector available at port 9811

► For a Node Agent:

SOAP connector available at port 8878

RMI Connector available at port 2809

For example, you can use the wsadmin interface with the following parameters:

```
wsadmin -conntype RMI -host localhost -port 2809
```

Alternatively, you can modify the wsadmin properties file with the application servers' or Node Agents' host and port as shown in Example 10-3.

*Example 10-3 The wsadmin properties file*

---

```
#-----
# The connectionType determines what connector is used.
# It can be SOAP or RMI.
# The default is SOAP.
#-----
#com.ibm.ws.scripting.connectionType=SOAP
com.ibm.ws.scripting.connectionType=RMI

#-----
# The port property determines what port is used when attempting
# a connection.
# The default SOAP port for a single-server installation is 8880
#-----
#com.ibm.ws.scripting.port=8880
#com.ibm.ws.scripting.port=8879
com.ibm.ws.scripting.port=2809
#-----
# The host property determines what host is used when attempting
# a connection.
# The default value is localhost.
#-----
com.ibm.ws.scripting.host=localhost
#com.ibm.ws.scripting.host=ws-coral
```

---

However, any change made to the Node Agent or the application server will be lost when the Node Agent synchronizes its repository with the master repository once the Deployment Manager comes up again.

## 10.5 Enhancing Deployment Manager high availability

There are several options to enhance the availability of the Deployment Manager, such as using clustering software and hardware.

### 10.5.1 Add Deployment Manager to OS daemon service

As a first step in making HA provisions for the Deployment Manager, you should make use of the platform specific mechanisms for providing operating system process monitoring and restart.

When installing WebSphere Network Deployment on the Windows platform, it is recommended that you select the option of adding the Deployment Manager as a Windows service. If you did not select this option during installation, then you can do so afterwards by using the **WASService** command (located in the <install\_root>\WebSphere\Deployment Manager\bin directory).

For Unix you can add an entry in the inittab file, as an example on AIX, you can do so by adding the following entry in the inittab file:

```
was:2:once:/usr/WebSphere/DeploymentManager/bin/rc.was >dev/console 2>&1
```

For Linux, you can use the **chkconfig** command.

Once you have added the Deployment Manager server as an operating system monitored process, the operating system will restart it automatically when the process terminates abnormally. This will minimize the impact of an outage due to a process failure.

Doing so will not prevent the failure of the Deployment Manager due to any network, disk, operating system, JVM, and host machine failures.

### 10.5.2 Make Deployment Manager highly available using clustering software and hardware

As noted before, the Deployment Manager server is a SPOF in WebSphere V5. As an option for 24x7 requirements, we discuss in the following sections how to make it highly available with clustering software and hardware.

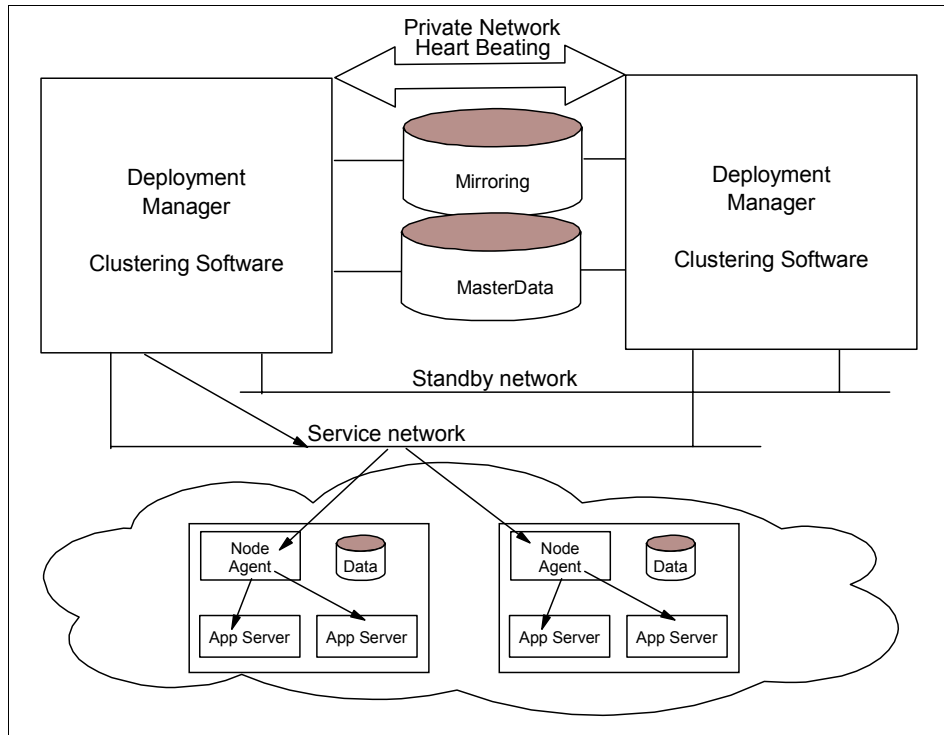


Figure 10-5 Clustered Deployment Manager for high availability

As shown in Figure 10-5, a software and hardware system includes HA clustering software (such as HACMP, VCS, ServiceGuard, SCS, and MSCS), a shared disk array subsystem, two host machines, a private service network, public primary network and optionally a backup network on different LANs that connect to other WebSphere nodes and other resources. The Deployment Manager is installed onto the shared disk and is configured in an HA resource group that includes the server hosting the Deployment Manager.

If a failure occurs in the Deployment Manager process, operating system, JVM, network, disk subsystem, repository data, and host machine, the heartbeat system will detect these failures and initiate a failover by:

- ▶ Stopping the Deployment Manager server process in the failed host
- ▶ Releasing the shared disk and other resources from this node
- ▶ Removing the service IP address from the failed node
- ▶ Mounting the shared disk to the other healthy node
- ▶ Adding the same service IP address to the adopted node
- ▶ Starting the Deployment Manager server

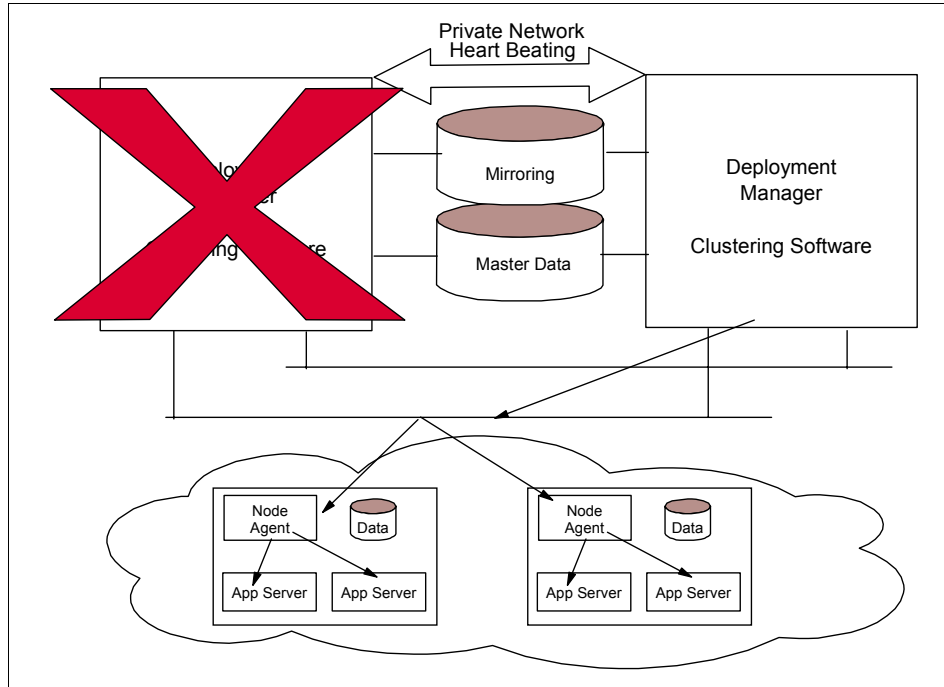


Figure 10-6 Failover of Deployment Manager with the assistance of HACMP

Depending on the hardware clustering product and configuration, you may need to make your Deployment Manager use the virtual (cluster) host name rather than the actual host name. Additionally, during our tests, we sometimes received an HACMP “rc207” return code that indicated a failure to stop the Deployment Manager and to release the shared disk resource. This happens because the Deployment Manager stop operation is initiated through asynchronous JMX messaging (wsadmin in our case). In this situation, the disk resource is still owned and held by this process. Adding a delay to the failover configuration script (as shown in Example 10-4) will allow the stop operation to finish completely so that the disk can be released before the shared disk resource is switched to the other node.

*Example 10-4 Add delay after stopManager command*

```
#/usr/bin/ksh
/HA/WebSphere/DeploymentManager/bin/stopManager.sh
sleep 60
```



**Note:** You might need some testing to find the right sleep time for your environment. If the sleep time is too short, you will still receive the rc207 error. If it is too long, the Deployment Manager failover will simply take a bit longer.

## HACMP configuration

Follow these steps to configure and install all necessary software:

### 1. Install HACMP:

- Before you configure HACMP, network adapters must be defined, the AIX operating system must be updated, and you must grant clustering nodes permission to access one another. Modify the following configuration files: `/etc/netsvc.conf`, `/etc/hosts`, and `/.rhosts`.

Make sure that each node's service adapters and boot addresses are listed in the `/.rhosts` file on each cluster node so that the `/usr/sbin/cluster/utilities/clruncmd` command and the `/usr/sbin/cluster/godm` command can run.

- Use `smit` to install HACMP 4.5, HACMP 4.3.1 or HACMP 4.4 or HACMP 4.4.1 or HACMP/ES 4.5 into both nodes. For installation details, see the *HACMP for AIX Installation Guide* at:

[http://www-1.ibm.com/servers/eserver/pseries/library/hacmp\\_docs.html](http://www-1.ibm.com/servers/eserver/pseries/library/hacmp_docs.html)

You can also install HACMP after you configure the network adapter and shared disk subsystem.

### 2. Public network configuration:

The public network is used to service client requests (WebSphere Deployment Manager management, WebSphere Application Server, applications, LDAP requests, for example). We defined two TCP/IP public networks in our configuration. The public network consists of a service/boot adapter and any standby adapters. It is recommended that you use one or more standby adapters. Define standby IP addresses and boot IP addresses. For each adapter, use `smit mktcpip` to define the IP label, IP address, and network mask. HACMP will define the service IP address. Since this configuration process also changes the host name, configure the adapter with the desired default host name last. Then use `smit chinnet` to change service adapters so that they will boot from the boot IP addresses. Check your configuration using `lsdev -Cc if`. Finally, try to ping the nodes to test the public TCP/IP connections.

### 3. Private network configuration:

A private network is used by the cluster manager for the heartbeat message; and can be implemented as a serial network in an HACMP cluster. A serial network allows the cluster manager to continuously exchange keep-alive

packets, should the TCP/IP-based subsystem, networks, or network adapters fail. The private network can be either a raw RS232 serial line, a target mode SCSI, or a target mode SSA loop. We used an HACMP for AIX serial line (a null-model, serial to serial cable) to connect the nodes. Use **smitty** to create the TTY device. After creating the TTY device on both nodes, test the communication over the serial line by entering the command **stty </dev/ttyx** on both nodes (where **/dev/ttyx** is the newly added TTY device). Both nodes should display their TTY settings and return to prompt if the serial line is okay. After testing, define the RS232 serial line to HACMP for AIX.

4. Shared disk array installation and LVG configuration:
  - The administrative data, application data, session data, LDAP data, log files and other file systems that need to be highly available are stored in the shared disks that use RAID technologies or are mirrored to protect data. The shared disk array must be connected to both nodes with at least two paths to eliminate the single point of failure. We used IBM 7133 Serial Storage Architecture (SSA) Disk Subsystem.
  - You can either configure the shared volume group to be concurrent access or non-concurrent access. A non-concurrent access environment typically uses journaled file systems to manage data, while concurrent access environments use raw logical volumes. There is a graphical interface called TaskGuide to simplify the task of creating a shared volume group within an HACMP cluster configuration. In Version 4.4, the TaskGuide has been enhanced to automatically create a JFS log and display the physical location of available disks. After one node has been configured, import volume groups to the other node by using **smit importvg**.
5. WebSphere Network Deployment (or WebSphere Enterprise Edition) configuration and federation of all nodes:
  - For installation details, see the manuals for the appropriate product, the InfoCenter, or the redbook *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195. You can install the products either on local disks for each node or on the shared disk array.
  - As mentioned before, you need to use virtual host names for this setup. It is better to change to using virtual host names *before* you install WebSphere. Otherwise you may need to manually change the virtual host names in all WebSphere configuration XML files afterwards. (Doing so is not recommended nor is it supported at present, though a set of scripts for doing so are slated for Web delivery as part of WebSphere Application Server V5.1.1.)

**Important:** Although you can install the WebSphere code on each node's local disks, it is very important to keep all *shared data*, such as the configuration repository and log files, on the shared disk array. This makes sure that another node can access this data when the current Deployment Manager node fails.

You will need to install the binaries for WebSphere Network Deployment (and/or Enterprise Edition) twice:

- i. Mount the shared disk array to the first node, for example as /ha. Install WebSphere into /ha as root, so the installation path is /ha/WebSphere/DeploymentManager.
- ii. After installation, start the Deployment Manager service using **startManager** to make sure everything is OK.
- iii. Delete all /ha/WebSphere/DeploymentManager files.
- iv. Mount the shared disk array to the same mount point on the other node, which again is /ha. Install WebSphere into /ha on the second node.
- v. When the installation is completed on the second node, test the environment again (start the Deployment Manager using **startManager**).

Doing this makes sure that all nodes pick up the WebSphere packages correctly.

**Note:** If you choose to install the software onto local disks rather than on the shared disk array, then you must install the same software version on both systems!

- Use **addNode.sh** to federate your WebSphere Application Server nodes to your Deployment Manager. The Node Agent is created and started automatically during the federation operation.
  - Start a browser and point it to the virtual host. You should be able to access the Administrative Console.
6. Define the cluster topology and HACMP application servers.

Please note that an HACMP application server is different from a WebSphere Application Server. In our case, the HACMP application server is the WebSphere Failover Unit, for example, the Deployment Manager.

- The cluster topology is comprised of the cluster definition, cluster nodes, network adapters, and network modules. The cluster topology is defined

by entering information about each component in HACMP-specific ODM classes. These tasks can be performed using **smit hacmp**. For details, see the *HACMP for AIX Installation Guide* at:

[http://www-1.ibm.com/servers/eserver/pseries/library/hacmp\\_docs.html](http://www-1.ibm.com/servers/eserver/pseries/library/hacmp_docs.html)

- An “application server”, in HACMP or HACMP/ES, is a cluster resource that is made highly available by the HACMP or HACMP/ES software, for example, in this case the WebSphere Deployment Manager. Do not confuse this with WebSphere Application Server. Use **smit hacmp** to define the HACMP (HACMP/ES) application server with a name and its start script and stop script.
- Start and stop scripts for the WebSphere Deployment Manager Failover Unit or DMgrFU (HACMP application servers).

Our sample DMgrFU service start script is:

```
/ha/WebSphere/DeploymentManager/bin/startManager.sh
```

And our sample DMgrFU service stop script is:

```
/ha/WebSphere/DeploymentManager/bin/stopManager.sh
```

You must test each configuration part/script individually before you can bundle all of them as the Failover Unit.

Please note that the WebSphere Failover Unit includes the WebSphere Application Server, Node Agent(s), and DB2 client. You must failover all of these together as an integrated failover unit, an individual server failover will not work.

#### 7. Define and configure the resource groups:

- For HACMP and HACMP/ES to provide a highly available application server service, you need to define the service as a set of cluster-wide resources essential to uninterrupted processing. The resource group can have both hardware and software resources such as disks, volume groups, file systems, network addresses, and application servers themselves.
- The resource group is configured to have a particular kind of relationship with a set of nodes. There are three kinds of node relationships: cascading, concurrent access, or rotating. For the cascading resource group, setting the cascading without fallback (CWOFF) attribute will minimize the client failure time. We used this configuration in our tests. Use **smit** to configure resource groups and resources in each group. Finally, you need to synchronize cluster resources to send the information contained on the current node to the other node.

#### 8. Cluster verification:

- Use `/usr/sbin/cluster/daig/clverify` on one node to check that all cluster nodes agree on the cluster configuration and the assignment of HACMP for AIX resources. You can also use `smit hacmp` to verify the cluster. If all nodes do not agree on the cluster topology and you want to define the cluster as it is defined on the local node, you can force agreement of cluster topology onto all nodes by synchronizing the cluster configuration. Once the cluster verification is satisfactory, start the HACMP cluster services using `smit hacmp` on both nodes, and monitor the log file using `tail -f /tmp/hacmp.out`. Check the database processes.

9. Takeover verification:

- To test a failover, use `smit hacmp` to stop the cluster service with the takeover option. On the other node, enter the `tail -f /tmp/hacmp.out` command to watch the takeover activity.

Example 10-5 shows the complete configuration file for a successful Deployment Manager failover.

*Example 10-5 Deployment Manager failover configuration script*

---

```
Cluster Description of Cluster hacluster
Cluster ID: 99
There were 2 networks defined : haserial, tr157
There are 2 nodes in this cluster.

NODE hacmp1:
    This node has 2 service interface(s):

        Service Interface hacmp1-tty:
            IP address:      /dev/tty0
            Hardware Address:
            Network:         haserial
            Attribute:       serial
            Aliased Address?: False

        Service Interface hacmp1-tty has no boot interfaces.
        Service Interface hacmp1-tty has no standby interfaces.

Service Interface hacmp1s:
IP address:      10.2.157.68
Hardware Address:
Network:         tr157
Attribute:       public
Aliased Address?: False
Service Interface hacmp1s has 1 boot interfaces.
Boot (Alternate Service) Interface 1: hacmp1b
IP address:      10.2.157.207
```

```
Network:      tr157
Attribute:    public
Service Interface hacmp1s has 1 standby interfaces.
Standby Interface 1: hacmp1sb
IP address:   10.10.0.30
Network:      tr157
Attribute:    public
```

NODE hacmp2:

This node has 2 service interface(s):

```
Service Interface hacmp2-tty:
IP address:    /dev/tty0
Hardware Address:
Network:       haserial
Attribute:     serial
Aliased Address?:      False
```

Service Interface hacmp2-tty has no boot interfaces.  
Service Interface hacmp2-tty has no standby interfaces.

Service Interface hacmp2s:

```
IP address:    10.2.157.196
Hardware Address:
Network:       tr157
Attribute:     public
Aliased Address?:      False
```

Service Interface hacmp1s has 1 boot interfaces.

```
Boot (Alternate Service) Interface 1: hacmp1b
IP address:    10.2.157.207
Network:       tr157
Attribute:     public
```

Service Interface hacmp1s has 1 standby interfaces.

```
Standby Interface 1: hacmp1sb
IP address:    10.10.0.30
Network:       tr157
Attribute:     public
```

NODE hacmp2:

This node has 2 service interface(s):

```
Service Interface hacmp2-tty:
IP address:    /dev/tty0
Hardware Address:
Network:       haserial
Attribute:     serial
```

Aliased Address?: False

Service Interface hacmp2-tty has no boot interfaces.  
Service Interface hacmp2-tty has no standby interfaces.

Service Interface hacmp2s:  
IP address: 10.2.157.196  
Hardware Address:  
Network: tr157  
Attribute: public  
Aliased Address?: False  
Service Interface hacmp2-tty has no boot interfaces.  
Service Interface hacmp2-tty has no standby interfaces.

Service Interface hacmp2s:  
IP address: 10.2.157.196  
Hardware Address:  
Network: tr157  
Attribute: public  
Aliased Address?: False

Service Interface hacmp2s has 1 boot interfaces.  
Boot (Alternate Service) Interface 1: hacmp2b  
IP address: 10.2.157.206  
Network: tr157  
Attribute: public  
Service Interface hacmp2s has 1 standby interfaces.  
Standby Interface 1: hacmp2sb  
IP address: 10.10.0.40  
Network: tr157  
Attribute: public

Breakdown of network connections:

Connections to network haserial

Node hacmp1 is connected to network haserial by these interfaces:  
hacmp1-tty

Node hacmp2 is connected to network haserial by these interfaces:  
hacmp2-tty

Connections to network tr157

Node hacmp1 is connected to network tr157 by these interfaces:  
hacmp1b  
hacmp1s

hacmp1sb

Node hacmp2 is connected to network tr157 by these interfaces:

hacmp2b

hacmp2s

hacmp2sb

Resource Group Name	ND
Node Relationship	cascading
Participating Node Name(s)	hacmp1 hacmp2
Dynamic Node Priority	
Service IP Label	hacmp1s
Filesystems	/ha
Filesystems Consistency Check	fsck
Filesystems Recovery Method	sequential
Filesystems/Directories to be exported	
Filesystems to be NFS mounted	
Network For NFS Mount	
Volume Groups	havg
Concurrent Volume Groups	
Disks	
Connections Services	
Fast Connect Services	
Shared Tape Resources	
Application Servers	CellManager
Highly Available Communication Links	
Miscellaneous Data	
Auto Discover/Import of Volume Groups	true
Inactive Takeover	false
Cascading Without Fallback	true
9333 Disk Fencing	false
SSA Disk Fencing	false
Filesystems mounted before IP configured	true
Run Time Parameters:	
Node Name	hacmp1
Debug Level	high
Host uses NIS or Name Server	false
Format for hacmp.out	Standard
Node Name	hacmp2
Debug Level	high
Host uses NIS or Name Server	false
Format for hacmp.out	Standard

---





## **WebSphere Embedded JMS server and WebSphere MQ high availability**

In IBM WebSphere Application Server Network Deployment V5.1, you can create application server clusters for scalability and high availability as described in the previous chapters. However, WebSphere itself doesn't provide MDB and messaging high availability.

It is possible to make IBM WebSphere Application Server Network Deployment V5.1 messaging and MDBs highly available for local and remote queue manager failures and application server failures to ensure messaging is timely processed once and only once.

In this chapter, we discuss several HA approaches (hardware clustering and software clustering, MQ clustering, and a combination of all) for both the WebSphere Embedded JMS server and external WebSphere MQ.

The content of this chapter has not changed since the previous version of this redbook.

## 11.1 JMS support in IBM WebSphere Application Server Network Deployment V5.1

J2EE 1.3 supports Java Message Service (JMS) which includes two types of messaging: point-to-point and publish/subscribe. Message-driven beans (MDBs) use these services. The WebSphere Embedded JMS server of WebSphere V5 supports all messaging functions as described in J2EE 1.3.

In addition to the WebSphere Embedded JMS server, which is included in WebSphere V5.1, you can also use the full WebSphere MQ product. WebSphere MQ supports more functions than Embedded JMS, such as queue manager clustering. WebSphere MQ needs to be purchased separately.

Depending on your needs, you will configure the appropriate JMS provider as shown in Figure 11-1:

- ▶ Embedded WebSphere JMS provider
- ▶ External WebSphere MQ JMS provider
- ▶ Other generic JMS providers



Figure 11-1 Various JMS providers in WebSphere V5

For details on the various JMS providers and how to configure them, refer to Chapter 15 of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195. The following is a high-level description of the configuration steps needed:

1. First, set the scope to cell to make sure that clustered WebSphere Application Servers on several hosts can access the same source of messaging, as shown in Figure 11-2 on page 419.

A JMS provider enables asynchronous messaging based on the Java Messaging Service (JMS). It provides J2EE connection factories to create connections for specific JMS queue or topic destinations. WebSphere MQ JMS provider administrative objects are used to manage JMS resources for WebSphere MQ as the JMS provider. [i](#)

Scope: Cell=RALYAS4A\_default, Node=RALYAS4A\_default

<input checked="" type="radio"/> Cell	RALYAS4A_default	Use scope settings to limit the availability of resources to a particular cell, node, or server. When new items are created in this view, they will be created within the current scope.
<input type="radio"/> → Node	RALYAS4A_default	
<input type="radio"/> Server	server1	
<input type="button" value="Apply"/>		

2. Next, you need to configure the Connection Factories and destinations for your queues or topics, as shown in Figure 11-3. Here you define the location of your JMS servers (host and port).

Additional Properties	
<a href="#">WebSphere MQ Queue Connection Factories</a>	
<a href="#">WebSphere MQ Topic Connection Factories</a>	
<a href="#">WebSphere MQ Queue Destinations</a>	
<a href="#">WebSphere MQ Topic Destinations</a>	

3. In order for MDBs to use the defined JMS provider, you need to specify the Message Listener Service in the Additional Properties of your application servers, as shown in Figure 11-4 on page 420.

Additional Properties	
<a href="#">Transaction Service</a>	Specify settings for the Transaction Service, as well as manage active transaction locks.
<a href="#">Web Container</a>	Specify thread pool and dynamic cache settings for the container . Also, specify session manager settings such as persistence and tuning parameters, and HTTP transport settings.
<a href="#">EJB Container</a>	Specify cache and datasource information for the container.
<a href="#">Dynamic Cache Service</a>	Specify settings for the Dynamic Cache service of this server.
<a href="#">Logging and Tracing</a>	Specify Logging and Trace settings for this server.
<a href="#">Message Listener Service</a>	Configuration for the Message Listener Service.This service provides the Message Driven Bean (MDB) listening process, whereby MDBs are deployed against ListenerPorts that define the JMS destination to listen upon. These Listener Ports are defined within this service along with settings for its Thread Pool.
<a href="#">ORB Service</a>	Specify settings for the Object Request Broker Service.
<a href="#">Custom Properties</a>	Additional custom properties for this runtime component. Some components may make use of custom configuration properties which can be defined here.

*Figure 11-4 Define Message Listener Service for WebSphere Application Servers*

4. The last step is to define MDB listener ports in the Additional Properties of the Message Listener Service. When you assemble/install your MDBs, you need to use these listener ports. Set the initial state to Started to make sure the listener ports are also started when your application server is started.

The Embedded JMS server can be administered (on the Deployment Manager server) through scripting or the Administrative Console. Figure 11-5 on page 421 shows the basic WebSphere Network Deployment configuration using the WebSphere Embedded JMS server.

When the JMS server (queue manager) fails, no application server can access messages available for processing, and application clients cannot put messages into the queue. Therefore, the JMS server becomes an SPOF for IBM WebSphere Application Server Network Deployment V5.1.

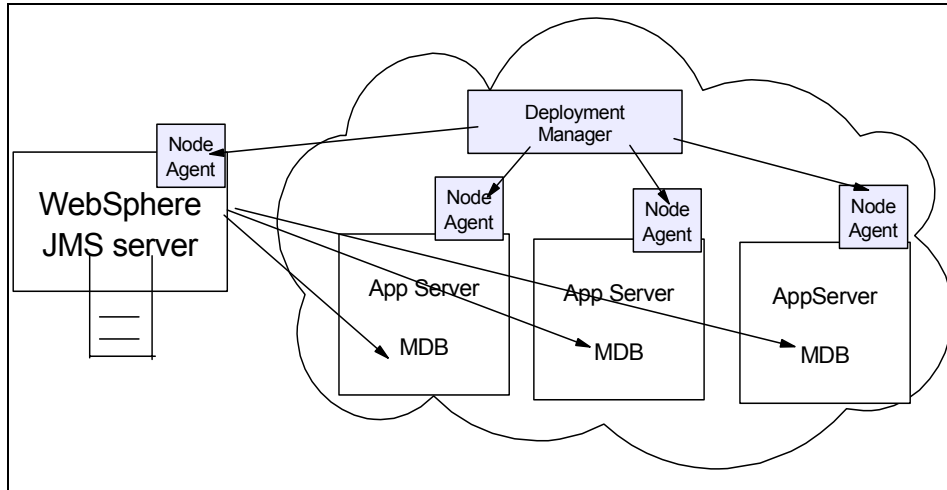


Figure 11-5 WebSphere Network Deployment and JMS server

There are four approaches to handling WebSphere JMS SPOF:

1. Use hardware-software clustering for WebSphere Embedded JMS server.
2. Use hardware-software clustering for WebSphere MQSeries® server.
3. Use WebSphere MQ built-in queue manager clustering.
4. Use a combination of hardware/software clustering and WebSphere MQ built-in queue manager clustering for best availability and scalability.

Approach 1 provides highly available IBM WebSphere Application Server Network Deployment V5.1 messaging. Approach 2 provides highly available WebSphere messaging, plus it is interoperable with non-JMS applications. Approach 3 provides scalable WebSphere messaging but messages in failed local queues may experience indefinite delays until the failed queue managers are restarted, so the timely processing of messages cannot be ensured. Approach 4 eliminates this problem and ensures that WebSphere messaging is highly available and scalable for time-critical applications.

## 11.2 Embedded JMS server high availability using clustering software

As shown in Figure 11-6 on page 422, clustering software (such as HACMP, ServiceGuard, VCS, SCS, MSCS), the network and hardware system can be used in conjunction with WebSphere Embedded JMS server in the IBM

WebSphere Application Server Network Deployment V5.1 Edition to build a highly available MOM (Message-oriented Middleware) system within a WebSphere V5 domain (cell).

Figure 11-7 on page 423 shows that such a system can tolerate any failure from the JMS server process, disk, operating system, host, and network. Therefore, this system removes the WebSphere Embedded JMS server SPOF. In addition, this system provides a transparent and single image view to application clients that need to put or retrieve messages. Refer to Chapter 12, “WebSphere data management high availability” on page 435 for information about how to configure the various clustering software.

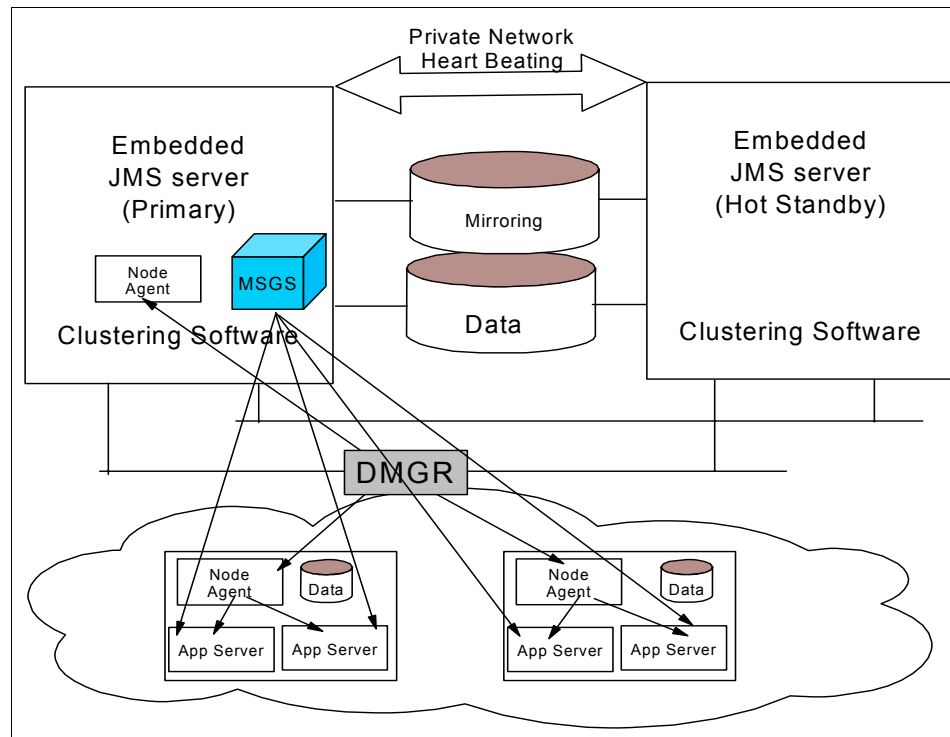


Figure 11-6 JMS server with clustering software

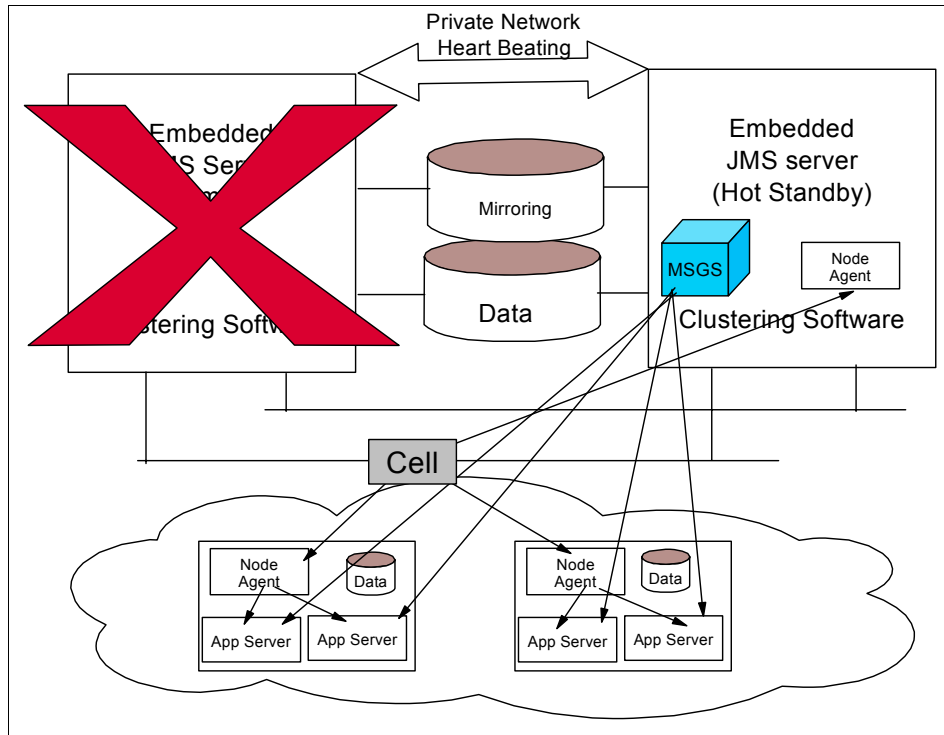


Figure 11-7 Embedded JMS server failover

WebSphere Application Server V5 is compatible with the J2EE 1.3 specifications for JMS. The WebSphere JMS programming environment can be accessed from the three containers (Web container, EJB container, and the client container). Applications running in these containers have access to JMS functions. Since the Embedded JMS server is part of WebSphere V5, it is administered by the same administrative server as WebSphere Application Servers. Therefore it is easy to install and configure and has the advantage of no cost for additional software.

There can be up to one Embedded WebSphere JMS server per node. However, different WebSphere Application Servers on different nodes within a cell that wish to communicate on a specific JMS queue must use a common Embedded WebSphere JMS server, as shown in Figure 11-5 on page 421. WebSphere Application Servers on remote nodes refer to the Embedded WebSphere JMS server by its WebSphere Application Server node.

The Embedded WebSphere JMS service is accessible only from WebSphere Application Server containers, and it is not interoperable with WebSphere MQ. The underlying technology for the Embedded WebSphere JMS server is provided by WebSphere MQ V5.3.0.1 for point-to-point messaging and

WebSphere MQ Event Broker 2.1 for publish/subscribe messaging. The concept behind the Embedded WebSphere JMS server is to provide easy access to JMS for J2EE programmers. Although the underlying technology is provided by WebSphere MQ, the Embedded WebSphere JMS server is not meant to replace external WebSphere MQ.

The WebSphere Embedded JMS server provides limited functions compared to WebSphere MQ. For example, it doesn't support QMgr to QMgr channel, message flows and message transformation. Also, it is not possible to communicate outside of WebSphere. In addition, it doesn't support database persistence.

## 11.3 WebSphere MQ HA using clustering software

We have discussed high availability clustering for the WebSphere Embedded JMS server in the previous section. This high availability solution can satisfy all J2EE messaging and HA requirements. Also, the Embedded WebSphere JMS server is as robust as WebSphere MQ.

However, as mentioned before, there are some situations when you need to use the more powerful external WebSphere MQ to extend messaging to heterogeneous environments or to more messaging application clients, as well as for message database persistency.

The queue managers in the Embedded WebSphere JMS servers are completely isolated; they have no channels either to other embedded JMS queue managers or to external WebSphere MQ queue managers.

The Embedded WebSphere JMS server cannot be used if an application requires access to heterogeneous non-JMS applications, or WebSphere MQ clustering, or the use of additional WebSphere MQ functions. WebSphere MQ can provide these functions.

The external WebSphere MQ cannot be installed in the same node with the Embedded WebSphere JMS server. The external queue manager must be Version 5.3 or above for XA coordination support.

Figure 11-8 on page 425 shows a system that uses clustering software (such as HACMP, ServiceGuard, SCS, VCS, and MSCS) and hardware for a highly available WebSphere MQ system to support J2EE applications in IBM WebSphere Application Server Network Deployment V5.1. If the queue manager process or network or primary host fails, the queue manager as well as the queue data and log will fail over to the hot standby machine with the same



service IP address. WebSphere Application Servers can continue to process messages without message losses.

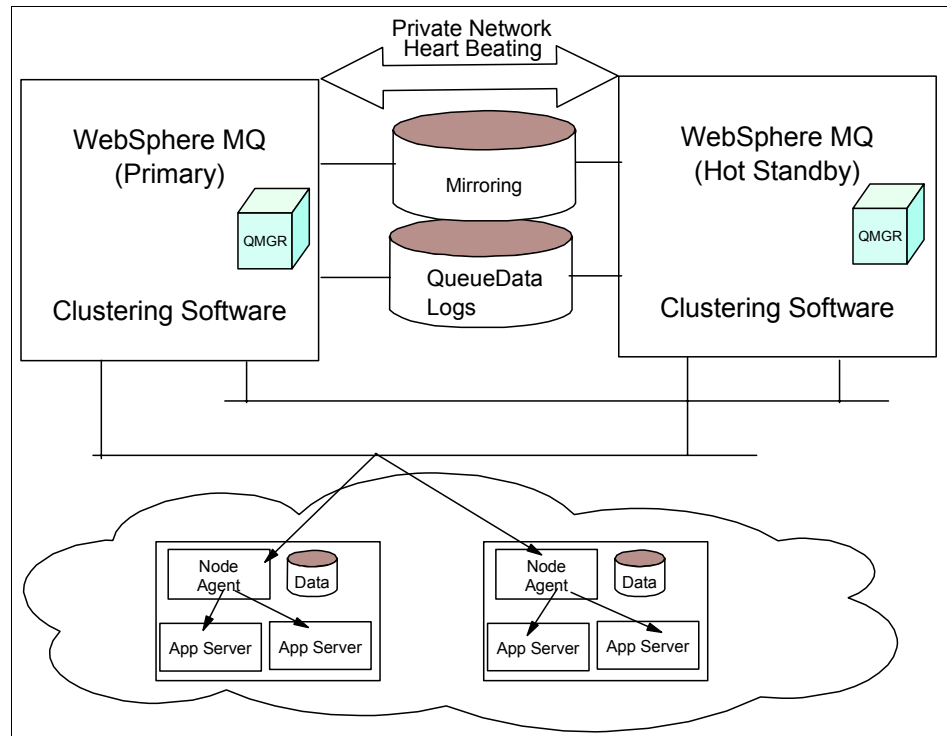


Figure 11-8 WebSphere MQSeries with clustering software

## 11.4 WebSphere MQ built-in queue manager clustering

The WebSphere MQ queue manager cluster concept reduces administration efforts and provides load balancing of messages across instances of clustered queues. Therefore, it also offers some level of high availability. However, local messages in the failed queue managers cannot be processed until the failed queue managers are restarted, because queue manager cannot be failed over with their states in a WebSphere MQ built-in cluster.

The WebSphere MQ cluster configuration is quite easy and straightforward. You have fewer channels to set up. WebSphere MQ clusters allow a group of queue managers to advertise the existence of some of their channels and queues to each other. The knowledge of these resources is shared among the queue managers and forms the cluster by including the queues and channels belonging to various queue managers. Because no single controlling entity is in charge of

the cluster, it is very robust, with no single point of failure. The collection of queue managers provides the service. As long as there are enough remaining queue managers and connections, the service is available. Failure of a network or failure of the destination queue manager manifests itself as a channel failure, with the communications channel performing its normal retry cycle. At the same time, any messages waiting on the `SYSTEM.CLUSTER.TRANSMIT.QUEUE` intended to move along the failed channel will be reprocessed and another destination found, if this is possible and appropriate. The communications channel between each pair of queue managers in the cluster is protected independently, without any global monitoring of the cluster as a whole.

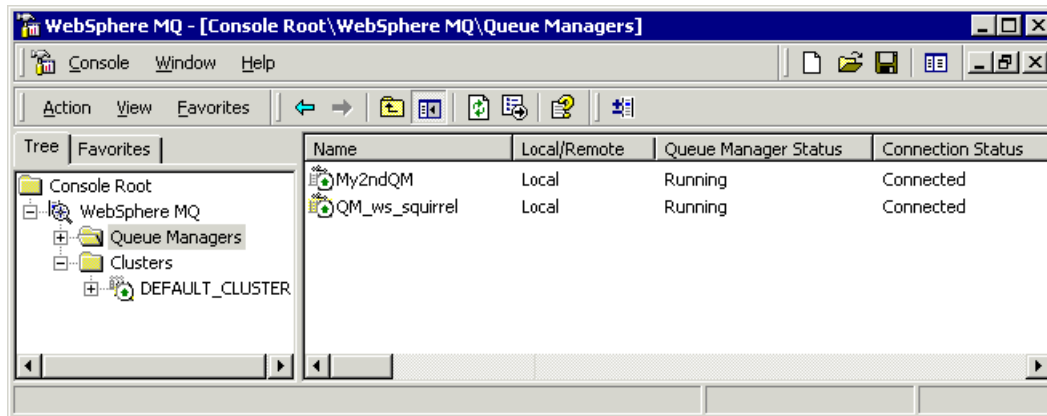


Figure 11-9 WebSphere MQ Explorer

The ability to define equivalent queues in a WebSphere MQ cluster, with identical names on several of the queue managers in the cluster, leads to increased system availability. Each queue manager runs equivalent instances of the applications that process the messages. Figure 11-10 on page 427 shows how a message put on one queue manager anywhere in the cluster is moved to other clustered queues and processed. If one of the queue managers fails, or the communication to it is suspended, it is temporarily excluded from the choice of destinations for the messages. This approach allows redundant servers to be completely independent of each other, thousands of miles apart if necessary. A problem in one server is less likely to affect the others.

Figure 11-10 on page 427 demonstrates the configuration and topology of a WebSphere Network Deployment V5 cluster (using horizontal and vertical scaling) with a WebSphere MQ cluster. This topology shows typical high availability characteristics for all variations of topologies. You can add more queue managers or application servers to the systems or add more physical boxes, but the essential high availability characteristics of WebSphere Network Deployment V5 and WebSphere MQ will not change.

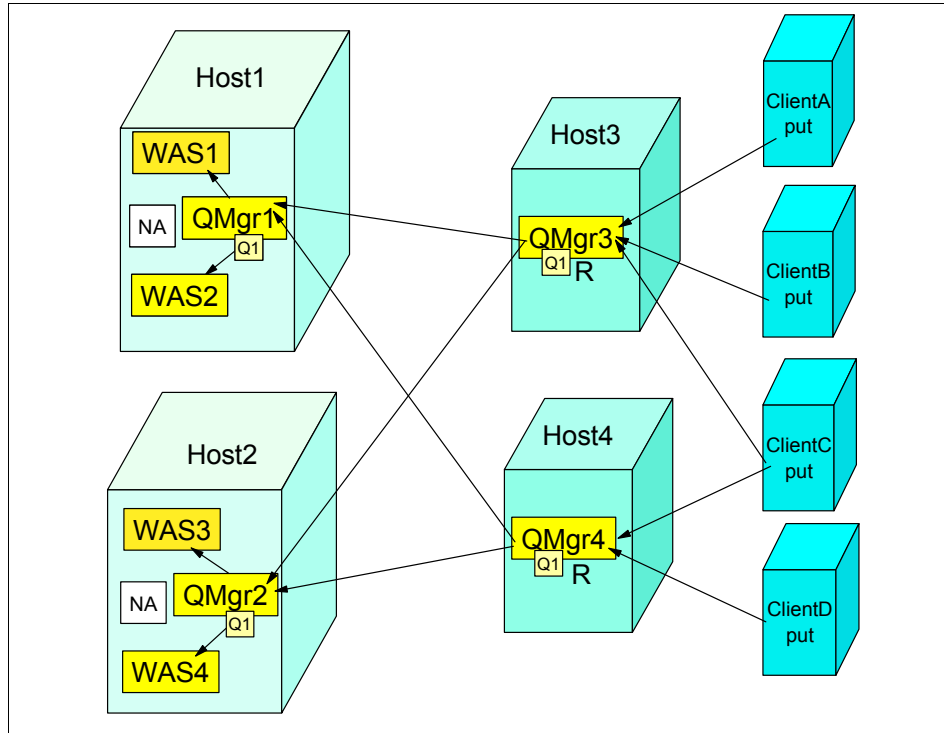


Figure 11-10 WebSphere MQ built-in queue manager cluster with WebSphere ND cluster

When the queue manager on Host2 fails, messages residing in the local queue will be trapped there, and the application servers WAS3 and WAS4 cannot process these trapped messages until the queue manager is restarted. The queue manager on Host1 is still functioning. It can handle new messages in its local queue that can be processed by the application servers on Host1.

If WAS1 fails, WAS2 continues to process messages from the local queue in Host1. If both WAS1 and WAS2 fail, the messages on QMgr1 will not be processed, and the messages on QMgr1 will increase to its maximum size or until WAS1 or WAS2 recover and are able to process them.

If Host1 fails or the network to Host1 is down, the messages will be distributed to QMgr2, which is located on Host2 and WAS3 and WAS4 will process them. If you use only WebSphere vertical scaling (no horizontal scaling), you will lose the protection for system or network failures.

If all messages are produced and consumed locally, this is a good high availability solution because of the messaging client redundancy. We have used this configuration for our Trade3 and Big3 test scenarios. However, locally

produced and consumed messages are not typical for real applications, since this reduces the value of asynchronous messaging. A messaging environment is meant to transfer messages over long distances, such as a credit card verification application. We have two ways to do it:

1. Message producers can use a session EJB to put messages on a queue. An MDB can consume these messages locally. Because the message producers (servlets and EJBs) are workload managed (message producer redundancy), clustered queues are a perfect HA solution for this case.
2. The clients send messages asynchronously to the queue on Host3, and the MDB processes the messages from the local queue (on Host1 or Host2), which is clustered with the queue on Host3. If the queue manager on Host3 fails, the clients cannot put messages into the queue and messages cannot be distributed to the local queues on Host1 and Host2. We can configure another queue manager on Host4 and allow clients to connect to either queue manager, but the clients have no server transparency (different IPs). In addition to the problem of stuck messages in failed local queues, the WebSphere MQ cluster doesn't provide a mechanism for triggering the failover of a queue manager.

WebSphere MQ clustered queues may be distributed across a wide geographical range. For an MQ cluster, at least two repository queue managers are needed for high availability. Also, at any site or access point, at least two producers/publishers or two consumers/subscribers are needed for high availability. We must have ways to either increase client chain redundancy, such as:

- ▶ Single client -> Servlet/EJB WLM-> multiple copies of the clients-> put/consume messages into/from the clustered queue.
- ▶ Alternatively, provide a single image of the queue manager to the client with hardware-based high availability for the queue manager.

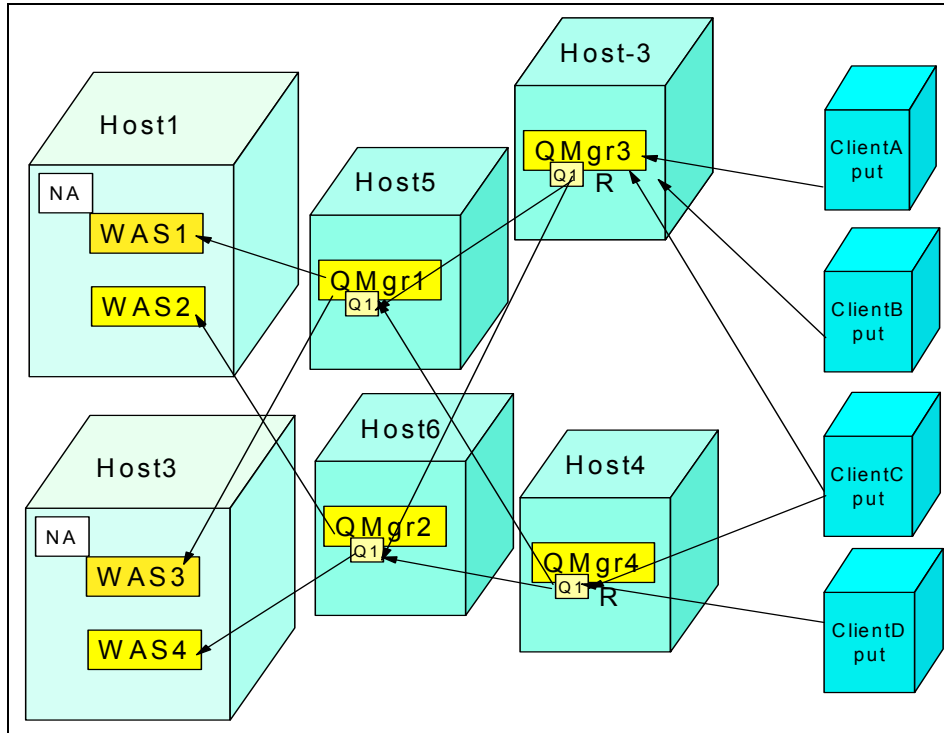


Figure 11-11 WebSphere ND V5 clustering with WebSphere MQ clustering

Network sprayers, such as the Load Balancer from the WebSphere Edge Components, can also be used to enhance WebSphere MQ cluster scalability and high availability as well as to provide a single access point for messaging application clients (the figure is similar to Figure 11-11, but with an HA Load Balancer in front of the WebSphere MQ clustered queue managers). However, messages trapped in the local queues of failed queue managers are still SPOFs since they cannot be processed until the failed queue managers are restarted.

The use of publish/subscribe in a WebSphere MQ cluster with cluster queues is not a practical configuration, because a subscriber has an identity, which in MQSeries terms is identified by the combination:

“Subscriber queue name + Queue manager name + Correlation ID”.

Since a JMS subscriber only receives publications that have its correlation ID, and no two correlation IDs are the same, a given publication message must go to its defined subscriber. This means that subscription queues should not be cluster queues when the hosting queue manager is part of a cluster. Figure 11-12 on page 430 shows a WebSphere cluster (using horizontal and vertical scaling) with listeners pointing to subscribe topics. It is very clear that the broker on Host3 is a

SPOF. We can use clustering techniques to make the publish/subscribe broker in Host3 highly available as discussed in the next section.

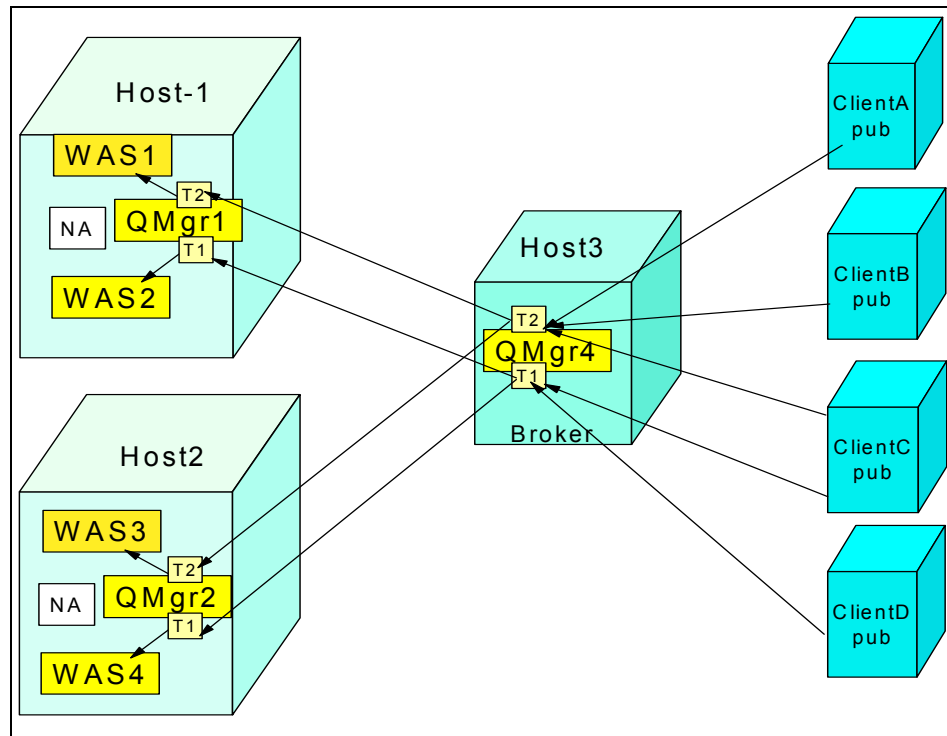


Figure 11-12 WebSphere horizontal/vertical cluster with MQ publish/subscribe

The best approach is to combine the WebSphere MQ built-in queue manager cluster and HACMP (or other kinds of clustering software). We discuss this scenario in the next section.

## 11.5 Combined approaches for MQ built-in queue manager cluster and HACMP clustering

A WebSphere MQ cluster in conjunction with a WebSphere Network Deployment V5 cluster gives you the capability to protect messages from being lost. Messages will be kept safe and eventually processed with assured delivery. However, in real life, eventually is usually not good enough. Messages are often time-critical and need to be processed sooner rather than later, so the systems that process them must be highly available.

Some messages represent actions on a high-value commodity, with a rapidly changing price. An example of this is stock trading, where a message stuck in a dead system is a liability. Messages representing trade or other instructions in these commodities must be processed quickly or the trader risks loss. Therefore, we also need solutions to ensure timely processing of time-critical messages by application servers in case of queue manager failures. The combined OS-clustering and MQ clustering can resolve this problem.

Figure 11-13 illustrates a system HA system that contains both the WebSphere MQSeries built-in queue cluster and HACMP-based clustering for messaging scalability and higher level availability.

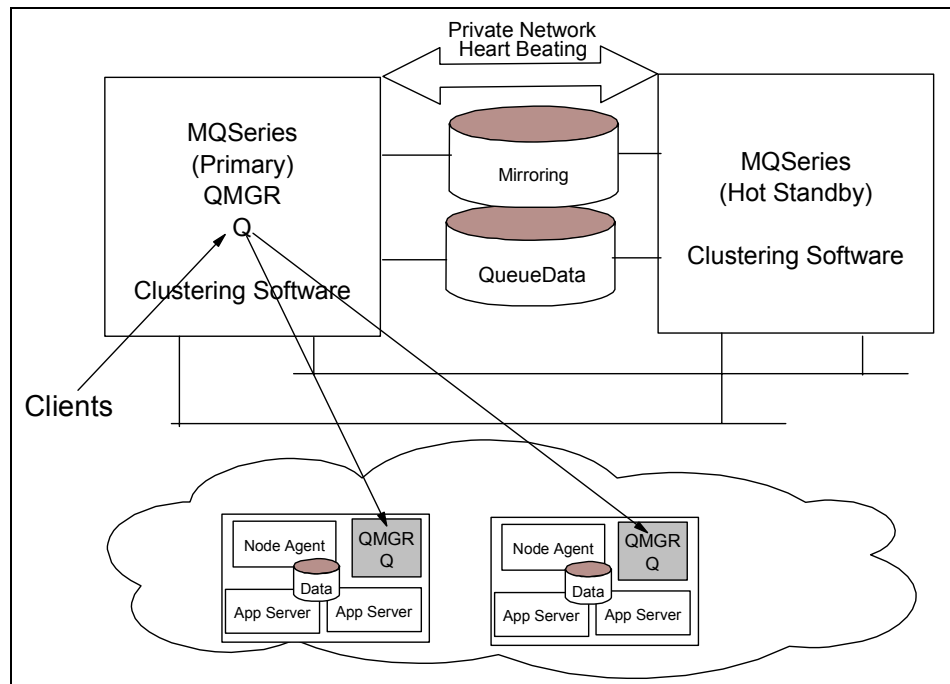


Figure 11-13 Combined WebSphere MQ Queue Manager clustering and hardware-based (HACMP, ...) clustering

The publish/subscribe broker is a SPOF. If the broker fails, no messages are available to be processed by the clustered application servers. We can use OS-clustering to ensure the high availability of the broker, as shown in Figure 11-14 on page 432.

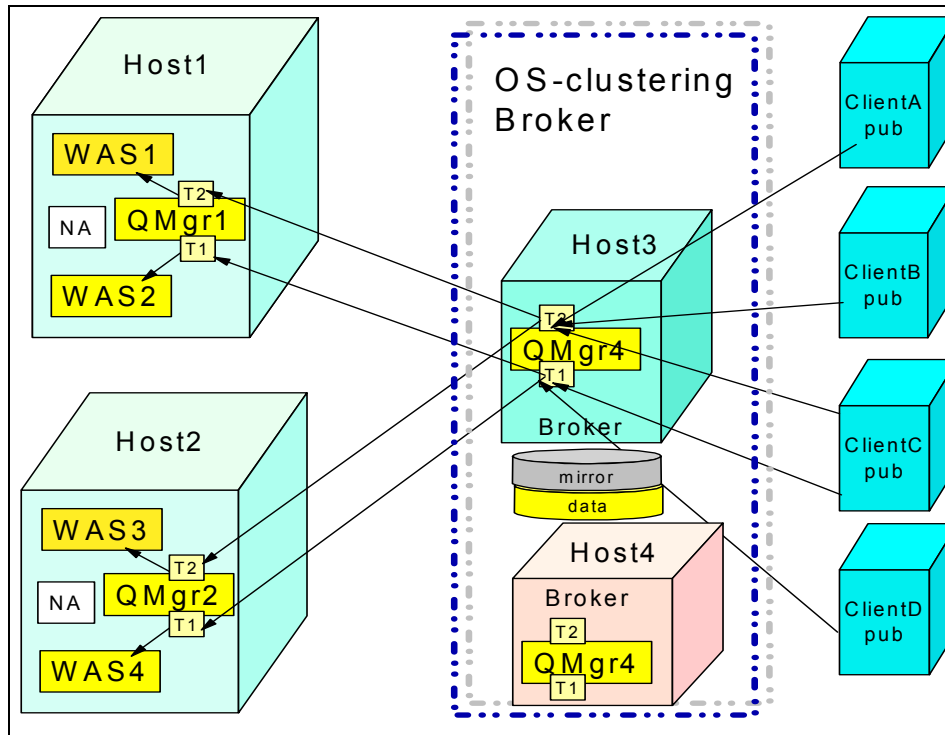


Figure 11-14 Clustering publish/subscribe broker

OS-clustering can be extended to all WebSphere MQ clustered managers, as shown in Figure 11-15 on page 433. All clustered queue managers and queues share a common storage that is RAID protected or a mirrored disk array. Queue and log data are stored in such highly available shared disk arrays. When any queue manager or host or network fails, the queue manager will be restarted automatically in another healthy host with the same data and log files so that the messages can be processed timely by the application servers.



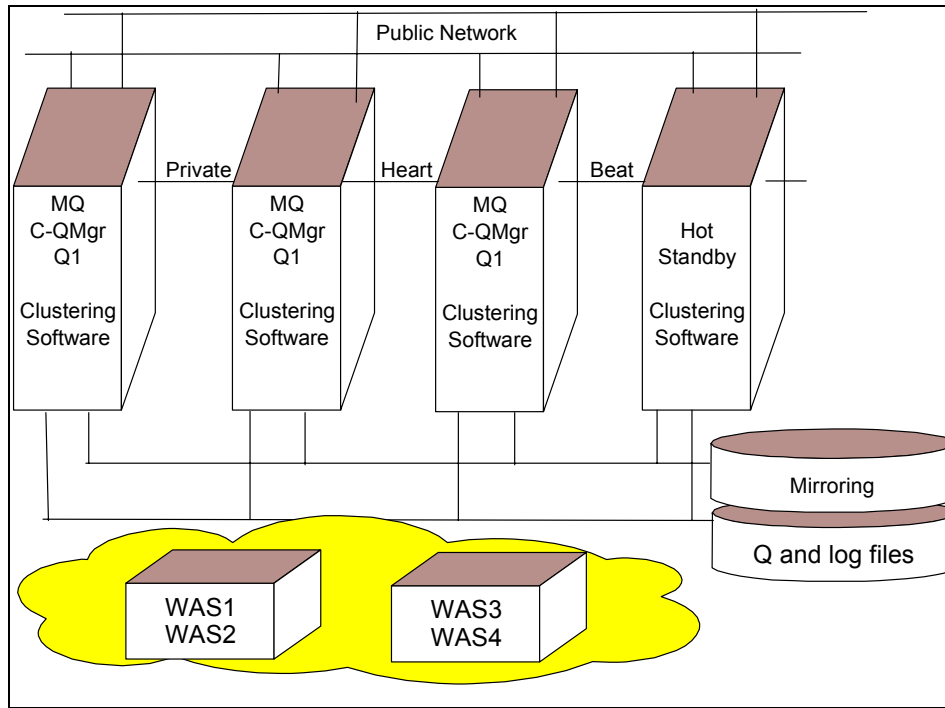


Figure 11-15 Combined: OS-clustering with MQ cluster

Various clustering techniques can be further extended and merged into a big cluster mix that ensures the highest availability and scalability, as shown in Figure 11-16 on page 434. Here, an IBM WebSphere Application Server Network Deployment V5.1 application server cluster, a WebSphere MQ cluster, and OS-clustering are coming together to ensure timely processing of messages, transaction log high availability, Deployment Manager high availability, and many other issues.

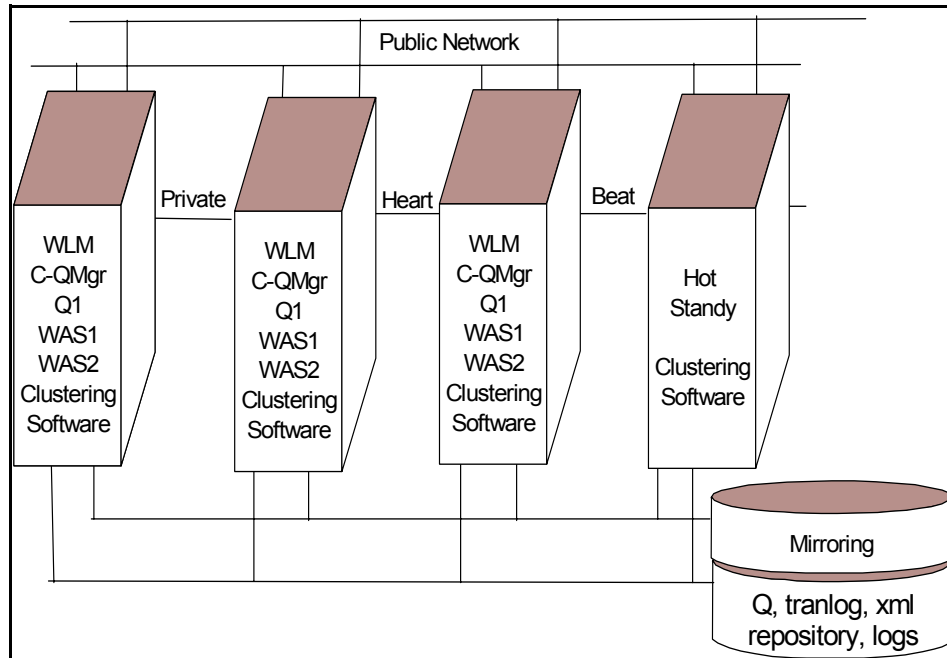


Figure 11-16 Combination of all clustering options



## WebSphere data management high availability

Data management is a central part of any e-business system. Data availability directly impacts WebSphere system availability. In this chapter we discuss various clustering techniques for WebSphere system, data management, high availability solutions, including:

- ▶ WebSphere with HACMP for Oracle
- ▶ WebSphere with HACMP for DB2
- ▶ WebSphere with MC/ServiceGuard for Oracle
- ▶ WebSphere with MC/ServiceGuard for DB2
- ▶ WebSphere with Sun Cluster for Oracle
- ▶ WebSphere with Sun Cluster for DB2
- ▶ WebSphere with VERITAS Cluster for Oracle
- ▶ WebSphere with VERITAS Cluster for DB2
- ▶ WebSphere with Microsoft Cluster for Oracle
- ▶ WebSphere with Microsoft Cluster for DB2

- ▶ WebSphere with Microsoft Cluster for SQL server
- ▶ WebSphere with Oracle Parallel Server, Transparent Application Failover (TAF)
- ▶ WebSphere with Oracle Parallel Server, Connection-Time Failover (CTF)
- ▶ WebSphere with Real Application Cluster, Transparent Application Failover
- ▶ WebSphere with Real Application Cluster, Connection-Time Failover
- ▶ WebSphere with DB2 Parallel Server guarded by HACMP cluster

The content of this chapter has not changed since the previous version of this redbook.

## 12.1 WebSphere with IBM HACMP

In this section, we look at integrating the WebSphere system with IBM HACMP. The topics discussed are:

- ▶ Introduction and considerations
- ▶ Hardware and software
- ▶ Setup and configuration
- ▶ Typical failover
- ▶ Initiating various failures and tests
- ▶ Tuning heartbeat and cluster parameters

### 12.1.1 Introduction and considerations

IBM HACMP for AIX is clustering software that can be used to build a highly available computing environment. It facilitates the detection of failures in hardware, software, and networks. It provides automatic failover of applications and resources from one system to another after a hardware, software, or network failure. This solution reduces downtime by removing single points of failure. We fully tested the WebSphere V5.1 end-to-end high availability solution, as shown in Figure 12-1 on page 439, which employed HACMP 4.5 and 4.4.x with DB2 7.2.1, Oracle 8i, and Oracle 9i for administrative repository, enterprise application data, persistent session data, log files, JMS providers, and LDAP data. We also tested HACMP/ES for these HA solutions. The major benefits of HACMP/ES arise from the technologies that underlie its heartbeat mechanism supplied in RISC System Cluster Technology (RSCT) and an application monitor.

The unit of failover in HACMP or HACMP/ES is a resource group. A resource group has to contain all the processes and resources needed to deliver a highly available service and ideally should contain only those processes and resources. HACMP for AIX software supports a maximum of up to 20 resource groups per cluster. Three types of resource groups are supported:

- ▶ **Cascading**, where a resource may be taken over by one or more nodes in a resource chain according to the takeover priority assigned to each node. The available node within the cluster with the highest priority will own the resource. You can also choose to set a flag so that a cascading resource will not fall back to a higher priority owner node when that node re-integrates with the cluster.
- ▶ **Rotating**, where a resource is associated with a group of nodes and rotates among these nodes. When a node fails, the next available node on its boot address and listed first in the resource chain will acquire the resource group.
- ▶ **Concurrent access**, where a resource that can be managed by the HACMP for AIX cluster lock manager may be simultaneously shared among multiple applications residing on different nodes.

The cascading resources type provides the best performance, because it ensures that an application is owned by a particular node whenever that node is active in the cluster. This ownership allows the node to cache data the application uses frequently. Rotating resources may minimize the downtime for failover.

In an HACMP cluster, there are public and private networks:

- ▶ A public network connects multiple nodes and allows clients to access these nodes.
- ▶ A private network is a point-to-point connection that links two or more nodes directly.

A network adapter (interface) connects a node to a network. A node typically is configured with at least two network interfaces for each network to which it connects:

- ▶ A service interface that handles cluster traffic
- ▶ One or more standby interfaces

The maximum number of network interfaces per node is 24. A service adapter must also have a boot address defined if IP address takeover is enabled. Using a standby adapter eliminates a network adapter as a single point of failure. IP address takeover is an AIX facility that allows one node to acquire the network address of another node in the cluster. To enable IP address takeover, a boot adapter address must be assigned to the service adapter on each cluster node. Nodes use the boot address after a system reboot and before the HACMP for

AIX software is started. When the HACMP for AIX software is started on a node, the node's service adapter is reconfigured to use the service address instead of the boot address. If the node should fail, a takeover node acquires the failed node's service address on its standby adapter, making failure transparent to clients using that specific service address. Place standby adapters on a separate subnet.

You can also set a two-node mutual takeover configuration, where both nodes are running different instances of the same application (application, administrative, session, and LDAP databases for example), and are standing by for one another. The takeover node must be aware of the location of specific control files and must be able to access them to perform startup after a failover. It is good practice to put the control file into the local file system of each node. Lay out the application and its data so that only the data resides on shared external disks. This arrangement can prevent software license violations, but it will also simplify failure recovery.

If one node is not available because of a software or hardware upgrade, application failure, or hardware failure, the database server on the failed node is automatically restarted on another node that has access to the same disks that contain the databases. The database on the failing node may be in a transactionally inconsistent state. When the database starts up on the surviving machine, it must go through a crash recovery phase to bring the database back to a consistent state. To maintain data integrity and transactional consistency, the database will not be completely available until the crash recovery has completed.

Failover is the process of having cluster resources move from an unavailable node to an available node. A related process, fallback (or fail back), occurs when resources are restored to the failed node after it is back online. Although you can set up automatic fallback once the primary node is available, we suggest that you use manual fallback to minimize the WebSphere service interruption.

In our WebSphere high availability solution, we eliminated single points of failure from the front to the back: network, HA firewalls, HA load balancers, multiple HTTP servers, HTTP plug-in redirection, multiple Web containers, multiple EJB servers, HA LDAP servers and database, HA administrative repository, HA enterprise application data, and HA persistent session data. We integrated, configured, and tested the WebSphere end-to-end high availability system shown in Figure 12-1 on page 439. It includes HA firewalls for the DMZ, HA WebSphere Edge Components' Load Balancer with a hot standby, multiple HTTP servers with the HTTP plug-in directing requests to Web containers, multiple application servers, HA LDAP servers and database, and finally HA databases with a hot standby database node.

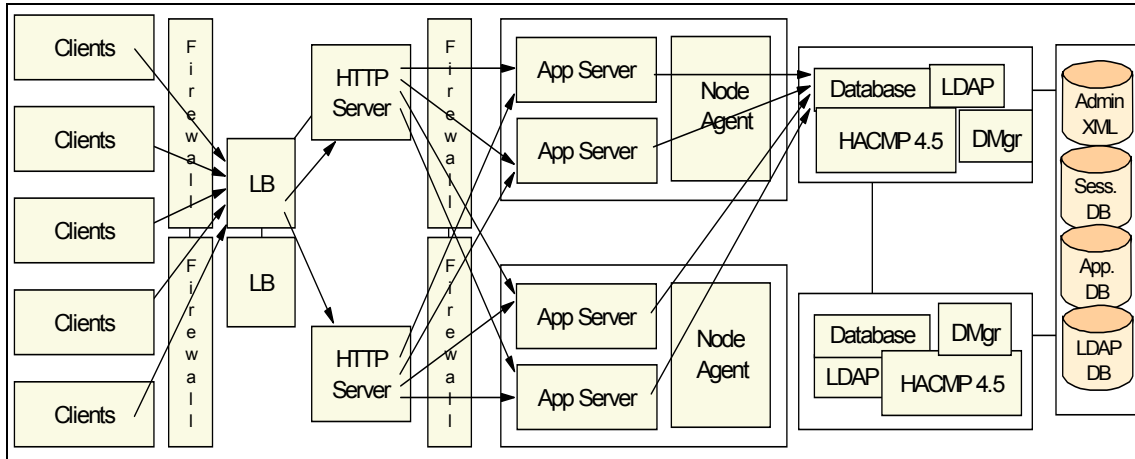


Figure 12-1 WebSphere and HACMP test environment

We tested four variations for this topology:

- ▶ A single WebSphere administrative domain (cell).
- ▶ Two WebSphere administrative domains (cells), with one application server and its cluster members on each node, for each domain (cell).
- ▶ Two WebSphere administrative domains (cells), but one for EJB servers and the other for servlet servers.
- ▶ Four WebSphere administrative domains (cells), two for EJB servers and two for servlet servers (for this test, we used two more machines).

We used Mercury LoadRunner to simulate customer site situations and to drive heavy load for two-day duration tests, and we initiated failures in all components until we achieved satisfactory availability results. We tested administrative actions, data source lookup, servlets, JSPs, session manager, session EJBs, CMP EJBs, and BMP EJBs individually for each failure point in each component. We also tested comprehensive benchmark applications such as Trade3, Big3, and others.

We initiated temporary failures in the network, Load Balancers, HTTP servers, application servers (both within a node and in different nodes), administrative servers (Node Agent and Deployment Manager), and databases. We determined that the total failed client requests were a very small percentage of total client requests, which depends on frequency of failure initiation. In WebSphere V5, the percentage of failed client requests has been significantly reduced by HA enhancements in many components such as WLM, session management,

naming, and system management. Since we have discussed the WebSphere configuration in the previous chapters, we will describe the integration of hardware clustering and WebSphere clustering and focus more on HA data management and its impacts on the whole WebSphere system.

## 12.1.2 Hardware and software

We used the following hardware and software in our test environment:

- ▶ IBM AIX machines (7)
- ▶ IBM PC (9)
- ▶ AIX 4.3.3 and 5.1
- ▶ AutoRaid disk array
- ▶ HACMP 4.3.1, HACMP 4.4, HACMP 4.5, or HACMP/ES 4.5
- ▶ IBM WebSphere Application Server Network Deployment V5.1
- ▶ DB2 UDB 7.2.1
- ▶ Oracle 8i and 9i
- ▶ Oracle Parallel Server (8i) and Oracle Real Application Cluster (9i)
- ▶ CheckPoint FireWall-1
- ▶ IBM hubs
- ▶ WebSphere Edge Components Load Balancer (2)

## 12.1.3 Setup and configuration

We set up and configured our test environment as follows:

1. Install HACMP 4.5 or HACMP 4.3.1 or HACMP 4.4 or HACMP 4.4.1 or HACMP/ES 4.5:
  - Before you configure HACMP, network adapters must be defined, the AIX operating system must be updated, and you must grant clustering nodes permission to access one another. Modify the following configuration files: `/etc/netsvc.conf`, `/etc/hosts`, and `/.rhosts`.  
  
Make sure that each node's service adapters and boot addresses are listed in the `/.rhosts` file on each cluster node so that the `/usr/sbin/cluster/utilities/clruncmd` command and the `/usr/sbin/cluster/godm` command can run.
  - Use `smit` to install HACMP 4.5, HACMP 4.3.1 or HACMP 4.4 or HACMP 4.4.1 or HACMP/ES 4.5 into both nodes. For installation details, see the *HACMP for AIX Installation Guide* at:  
[http://www-1.ibm.com/servers/eserver/pseries/library/hacmp\\_docs.html](http://www-1.ibm.com/servers/eserver/pseries/library/hacmp_docs.html)  
  
You can also install HACMP after you configure the network adapter and shared disk subsystem.



## 2. Service network configuration:

The public network is used to provide services to clients (WebSphere, applications, LDAP, for example). We defined two TCP/IP public networks in our configuration. The public network consists of a service/boot adapter and any standby adapters. It is recommended that you use one or more standby adapters. Define standby IP addresses and boot IP addresses. For each adapter, use **smit mktcpip** to define the IP label, IP address, and network mask. HACMP will define the service IP address. Since this configuration process also changes the host name, configure the adapter with the desired default host name last. Then use **smit chinnet** to change service adapters so that they will boot from the boot IP addresses. Check your configuration using **lsdev -Cc if**. Finally, try to ping nodes to test the public TCP/IP connections.

## 3. Serial network configuration:

A serial network is needed for the heartbeat message; it is a serial network in an HACMP cluster. A serial network allows the cluster manager to continuously exchange keep-alive packets, should the TCP/IP-based subsystem, networks, or network adapters fail. The private network can be either a raw RS232 serial line, a target mode SCSI, or a target mode SSA loop. The HACMP for AIX serial line (a null-model, serial to serial cable) was used to connect the nodes. Use **smitty** to create the TTY device. After creating the TTY device on both nodes, test the communication over the serial line by entering the command **stty </dev/ttyx** on both nodes (where **/dev/ttyx** is the newly added TTY device). Both nodes should display their TTY settings and return to prompt if the serial line is okay. After testing, define the RS232 serial line to HACMP for AIX.

## 4. Shared disk array installation and LVG configuration:

- The administrative data, application data, session data, LDAP data, log files and other file systems that need to be highly available are stored in the shared disks that use RAID technologies or are mirrored to protect data. The shared disk array must be connected to both nodes with at least two paths to eliminate the single point of failure. We used IBM 7133 Serial Storage Architecture (SSA) Disk Subsystem.
- You can either configure the shared volume group to be concurrent access or non-concurrent access. A non-concurrent access environment typically uses journaled file systems to manage data, while concurrent access environments use raw logical volumes. There is a graphical interface called TaskGuide to simplify the task of creating a shared volume group within an HACMP cluster configuration. In Version 4.4, the TaskGuide has been enhanced to automatically create a JFS log and display the physical location of available disks. After one node has been configured, import volume groups to the other node by using **smit importvg**.

5. DB2 or Oracle and LDAP installation, configuration, instance and database creation:
  - For installation details, see the manuals for these products. You can install these products either on the disks in both nodes or on the shared disk. But you must keep all the shared data such as database files, transaction log files, and other important files on the shared disk array so that another node can access this data when the current node fails. We chose to install these products on each node. In this case, you must install the same version of the products on both nodes.
  - Create DB2 or Oracle instances on the shared disk subsystem. We created two DB2 instances for the WebSphere application database and session database. On the other test with Oracle, we also created two Oracle instances for the WebSphere application database and session database. You may need to change the `appheapsz` for the created DB2 database, or the cursor parameters for Oracle. See the WebSphere installation guide for details. See 12.11, “One instance for all WebSphere databases?” on page 493 to decide whether one instance or multiple instances must be used for these databases.
  - If “thick” database clients are used, install the database clients on your WebSphere nodes and configure the database clients to connect to the database server. For example, install DB2 clients on all WebSphere nodes and catalog the remote node and database server.
6. Define the cluster topology and HACMP application servers:
  - The cluster topology is comprised of cluster definition, cluster nodes, network adapters, and network modules. The cluster topology is defined by entering information about each component in HACMP-specific ODM classes. These tasks can be performed using `smit hacmp`. For details, see the *HACMP for AIX Installation Guide* at:  
[http://www-1.ibm.com/servers/eserver/pseries/library/hacmp\\_docs.html](http://www-1.ibm.com/servers/eserver/pseries/library/hacmp_docs.html)
  - An “application server”, in HACMP or HACMP/ES, is a cluster resource that is made highly available by the HACMP or HACMP/ES software, for example in our case DB2 databases, Oracle databases, and LDAP servers. Do not confuse this with WebSphere Application Server. Use `smit hacmp` to define the HACMP (HACMP/ES) application server with a name and its start script and stop script.
  - Start and stop scripts for both DB2 and Oracle as the HACMP application servers.

Our sample DB2 service start script is:

```
db2start
```

And our sample DB2 service stop script is:

```
db2 force applications all
db2stop
```

For Oracle, our sample service start script is:

```
lsnrctl start
export SIDS="APP ADMIN SESSION"
for SID in $$SIDS ; do
    export ORACLE_SID=$SID
    echo "connect internal\nstartup\nquit" | svrmgrl
done
```

And our sample Oracle service stop script is:

```
export SIDS="APP ADMIN SESSION"
for SID in $$SIDS ; do
    export ORACLE_SID=$SID
    echo "connect internal\nshutdown\nquit" | svrmgrl
done
lsnrctl stop
```

You must be the DB2 or Oracle user to use the above scripts. Otherwise, you need to change users with the **su** - command.

#### 7. Define and configure the resource groups:

- For HACMP and HACMP/ES to provide a highly available application server service, you need to define the service as a set of cluster-wide resources essential to uninterrupted processing. The resource group can have both hardware and software resources such as disks, volume groups, file systems, network addresses, and application servers themselves.
- The resource group is configured to have a particular kind of relationship with a set of nodes. There are three kinds of node relationships: cascading, concurrent access, or rotating. For the cascading resource group, setting the cascading without fallback (CWOFF) attribute will minimize the client failure time. We used this configuration in our tests. Use **smit** to configure resource groups and resources in each group. Finally, you need to synchronize cluster resources to send the information contained on the current node to the other node.

#### 8. Cluster verification:

- Use **/usr/sbin/cluster/daig/clverify** on one node to check that all cluster nodes agree on the cluster configuration and the assignment of HACMP for AIX resources. You can also use **smit hacmp** to verify the cluster. If all nodes do not agree on the cluster topology and you want to define the cluster as it is defined on the local node, you can force agreement of cluster topology onto all nodes by synchronizing the cluster

configuration. Once the cluster verification is satisfactory, start the HACMP cluster services using **smit hacmp** on both nodes, and monitor the log file using **tail -f /tmp/hacmp.out**. Check the database processes using **ps -ef | grep db2** or **ora**.

9. Takeover verification:

- To test a failover, use **smit hacmp** to stop the cluster service with the takeover option. On the other node, enter the **tail -f /tmp/hacmp.out** command to watch the takeover activity.

## 12.1.4 Typical failover

Figure 12-2 on page 445 and Figure 12-3 on page 446 show a typical HACMP-controlled failover. A hot standby with cascading resource groups is configured. Before the failover occurs, the shared disk array is mounted on the primary node, and database and LDAP processes are running on the primary node. WebSphere and applications access the databases through the service IP address. In order to enable this service IP address takeover, you need to define a boot IP address to this adapter.

If the primary node fails, HACMP will detect the failure, run the stop scripts and release the disk volume groups and other shared resources held by the primary node. By means of the heartbeat mechanism between two nodes, the standby node will take over the service IP address, mount the shared disk array, take over other resources, and run the start script. Even though the service runs in the different computer nodes, the same IP address is used to access the database server before and after failover. Therefore, such failover is transparent to clients, since the clients do not need to use different IP addresses for accessing the servers. However, the in-flight connections become invalid during the failover. Therefore, WebSphere must provide the capacity to handle such situations. Support for this feature began in WebSphere V3.02 with HACMP 4.3.1. This feature was improved in WebSphere V5 (and later releases of WebSphere V3.5.x and WebSphere V4) to minimize failures.

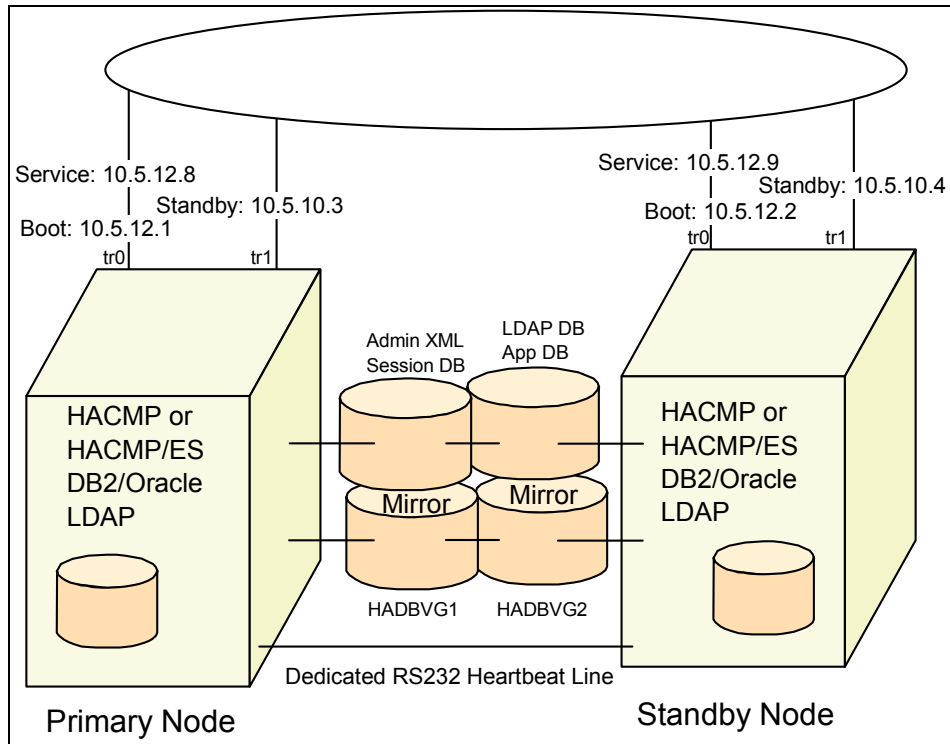


Figure 12-2 HACMP cluster configuration for WebSphere

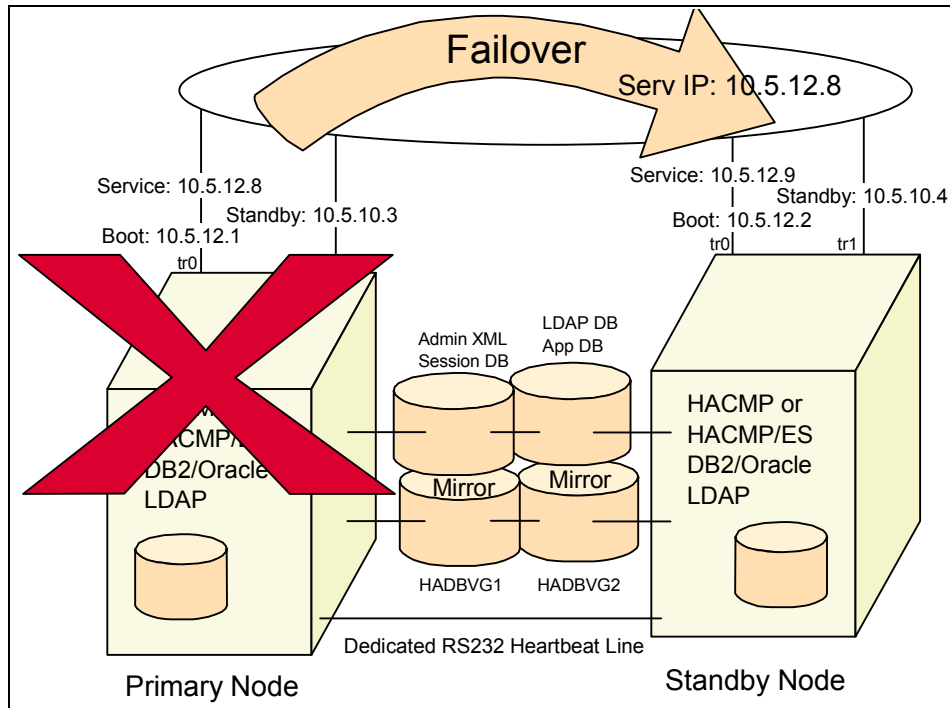


Figure 12-3 HACMP cluster after failover

### 12.1.5 Initiating various failures and tests

To start or stop the HACMP cluster service in a single node, you can issue the commands:

```
smit clstart
smit clstop
```

HACMP also provides a central administrative utility, so you can use the C-SPOC (Cluster Single Point of Control) utility to start the whole HACMP cluster by issuing the commands:

```
smit cl_clstart.dialog
smit cl_clstop.dialog
```

There are several places to look up log information:

- ▶ `/usr/adm/cluster.log` provides a high-level view of the current cluster status. It is a good place to look first when diagnosing a cluster problem.
- ▶ `/tmp/hacmp.out` is the primary source of information when investigating a problem.

You can also configure a remote machine to connect to the HACMP cluster and use `clstat` (or `xc1stat`) to monitor and test the HACMP cluster.

We initiated various failures at all the possible failure points in our WebSphere test topologies.

For the HACMP cluster, we initiated the failures by:

- ▶ Starting/stopping HACMP cluster services
- ▶ Powering down the database server node
- ▶ Disconnecting the network cable
- ▶ Killing the database processes
- ▶ Stopping the database servers

For the other parts in our WebSphere test topologies, we initiated failures by:

- ▶ Killing HTTP, application, administrative (Node Agent and Deployment Manager), and LDAP server processes
- ▶ Stopping HTTP, application, administrative (Node Agent and Deployment Manager), and LDAP server processes
- ▶ Disconnecting the network cable to each node
- ▶ Powering off each node one at a time

We initiated various kinds of failures in our test topologies, and we counted the failed client requests versus the total client requests. The failure rate was very small (<1%), depending on the frequency of initiation of various failures, HACMP or HACMP/ES detection and recovery time, and network characteristics. We found that WebSphere process failures were instantaneously recovered through the WebSphere WLM mechanism and contributed very little to failed requests. However, database recovery took minutes to complete and contributed the most part to the failed requests. Therefore, in the next section, we discuss how to tune the clustering parameters.

### 12.1.6 Tuning heartbeat and cluster parameters

We can work on both WebSphere connection techniques and tuning HACMP or HACMP/ES parameters to enhance system performance. WebSphere V5 has significantly improved failure-handling techniques compared to earlier versions of WebSphere. HACMP 4.5 provides easier and greater control over several tuning parameters that affect the cluster's performance. Setting these tuning parameters correctly ensures throughput and adjusting the HACMP failure detection rate can help avoid failures caused by heavy network traffic.

Some options are:

- ▶ Improving WebSphere connection pool techniques to minimize the failures when reconnecting after a failover. We fully tested WebSphere V5.x, WebSphere V4.x, WebSphere V3.5.x, and WebSphere V3.02 with HACMP 4.5, HACMP 4.3.1, HACMP 4.4, HACMP 4.4.1, and HACMP/ES for both DB2 and Oracle. The entire end-to-end high availability system performed very well. WebSphere V5 and WebSphere V3.5.3 and later versions have significantly improved connection pool techniques to minimize the failures during an HACMP failover, compared to WebSphere V3.02 and WebSphere V3.5.2 tested with HACMP 4.3.1.
- ▶ Adjusting high and low watermarks for I/O pacing. The default value is 33 -24.
- ▶ Adjusting syncd frequency rate. The default value is 10.
- ▶ Adjusting HACMP failure detection rate. There are two parameters that control the HACMP failure detection rate are:
  - HACMP cycles to failure: the number of heartbeats that must be missed before detecting a failure.
  - HACMP heartbeat rate: the number of microseconds between heartbeats.  
For example, if heartbeat rate=1 second and cycles=10, the failure detection rate would be 10 seconds. Faster heartbeat rates may lead to false failure detection, particularly on busy networks. Please note that the new values will become active the next time the cluster services are started.
- ▶ AIX deadman switch timeout. If HACMP for AIX cannot get enough CPU resource to send heartbeats on IP and serial networks, other nodes in the cluster will assume the node has failed, and initiate takeover of the node's resources. In order to ensure a lean takeover, the deadman switch crashes the busy node if it is not reset within a given time period.

## 12.2 WebSphere with MC/ServiceGuard

In this section, we look at integrating the WebSphere system with MC/ServiceGuard. The topics discussed are:

- ▶ Introduction and considerations
- ▶ Hardware and software
- ▶ Setup and configuration
- ▶ Typical failover
- ▶ Initiating various failures and tests
- ▶ Tuning heartbeat and cluster parameters



## 12.2.1 Introduction and considerations

MC/ServiceGuard, or Multi-Computer/ServiceGuard, allows you to create high availability clusters of HP 9000 series computers to compose a high availability computer system that allows databases and applications to continue in spite of a hardware, software, or network failure. This highly available system detects any fault through a heartbeat mechanism and starts the recovery process automatically. We put the WebSphere master administrative repository, enterprise application database, WebSphere session database, log files, JMS providers, LDAP database, and their processes under control of this MC/ServiceGuard system to protect users from software failures as well as from failures of system processing unit, disk, or local area network (LAN) components.

As shown in Figure 12-4, we integrated, configured, and tested WebSphere end-to-end high availability system including HA firewalls that create a DMZ zone, HA Load Balancer with a hot standby LB, multiple HTTP servers with HTTP plug-in that directs requests to Web containers, multiple application servers, HA JMS server, HA LDAP server, and HA databases with a hot standby database node. Such a system eliminates single points of failure in the network, firewall, load balancer, HTTP servers, application server, administrative servers, JMS server, database server, and LDAP server.

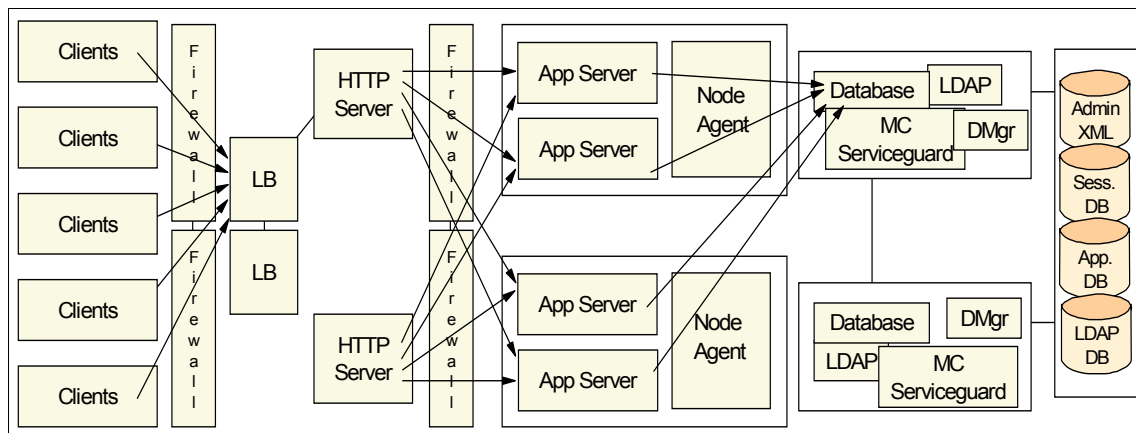


Figure 12-4 WebSphere and MC/ServiceGuard test environment

We tested four variations for this topology:

- ▶ A single WebSphere administrative domain (cell).
- ▶ Two WebSphere administrative domains (cells), each node containing one application server and its cluster members for each domain (cell).

- ▶ Two WebSphere administrative domains (cells), with one for EJB servers and the other for servlet servers.
- ▶ Four WebSphere administrative domains (cells), with two for EJB servers and two for servlet servers (for this test, we used two more machines).

We used Mercury LoadRunner to simulate customer site situations and to drive a heavy load for two-day duration tests, and we initiated failures in all components until we achieved satisfactory availability results. We tested administrative actions (Node Agent and Deployment Manager), servlets, JSPs, session manager, session EJBs, CMP EJBs, and BMP EJBs individually for each failure point in each component. We also tested comprehensive benchmark applications such as Trade3, Big3, and others.

We initiated temporary failures in the network, Network Dispatchers, HTTP servers, application servers (both within a node and in different nodes), administrative servers (Node Agent and Deployment Manager), and databases. We determined that the total failed client requests were a very small percentage of total client requests, which depends on frequency of failure initiation. Since we have discussed the WebSphere configuration in the previous chapters, we will focus more on HA data management and its impact on the whole WebSphere system.

## 12.2.2 Hardware and software

We used the following hardware and software in our test environment:

- ▶ HP 9000 K-class machines (4)
- ▶ IBM PC (9)
- ▶ IBM AIX machines (2)
- ▶ HP\_UX
- ▶ AutoRaid disk array
- ▶ MC/ServiceGuard A11.12 (2)
- ▶ WebSphere Application Server Network Deployment V5
- ▶ DB2 UDB 7.2.1
- ▶ Oracle 8i and 9i
- ▶ CheckPoint FireWall-1
- ▶ IBM hubs
- ▶ WebSphere Edge Components Load Balancer (2)

## 12.2.3 Setup and configuration

There are seven daemon processes associated with MC/ServiceGuard to assist configuration: clustering, syslog, LVM (logical volume management), cluster object manager, SBMP subagent, service assistant, and shared tape. The heartbeat mechanism is used to monitor node and database services and their

dependencies. We installed MC/ServiceGuard on two HP 9000 K-class machines and an AutoRAID disk array connected to both machines. We installed DB2 and Oracle on both machines and created instances on the shared disk array.

MC/ServiceGuard supports single-ended SCSI, Fast/Wide SCSI, and Fiber Channel disk interfaces. We used Fast/Wide SCSI interfaces to connect two nodes to an HP disk array. There are two ways to protect your data:

- ▶ Use disk mirroring. You can configure logical volumes using MirrorDisk/UX, so the members of each mirrored set contain exactly the same data. If one disk should fail, MirrorDisk/UX will automatically keep the data available by accessing the other mirror. To protect against SCSI bus failures, each copy of the data must be accessed by a separate SCSI bus. It is optional for you to mirror root disks since other nodes can take over the failed database process in case of failure of the original root disks.
- ▶ Use disk arrays using RAID levels and PV links. The array provides data redundancy for the disks. This protection needs to be combined with the use of redundant SCSI interfaces between each node and the array. Configured with PV links, the redundant interfaces protect against a single point of failure in the I/O channel. You can monitor disks through the event monitoring service.

For software failures, the database can be restarted on the same node or another node with minimum disruption. For failures of disk interfaces or other monitored resources, the database can be moved to another node. For a failure of the node itself, the database can be moved from a failed node to another node automatically. For failure of the LAN, MC/ServiceGuard switches to a standby LAN or moves the database to a standby node.

We need to have at least two heartbeat LANs in the MC/ServiceGuard cluster to keep the heartbeat message highly available. If you have only one heartbeat LAN, a dedicated serial heartbeat is required for two-node heartbeat communications. Redundancy is provided by the primary LAN and the dedicated LAN, which are both carrying the heartbeat. A dedicated heartbeat line will prevent a false diagnosis of heartbeat failure. We used a dedicated private Ethernet LAN for the heartbeat in addition to another heartbeat connection served by one of the public LANs, as shown in Figure 12-5 on page 455.

We set up and configured our test environment as follows:

1. Install MC/ServiceGuard software on each node with **swinstall** and choose the B3935DA package. For MC/ServiceGuard installation details, see the manual provided with MC/ServiceGuard software.

2. Configure and update each node for the MC/ServiceGuard cluster:

- a. Grant security permissions to both machines by adding entries into `/etc/cmclustercmclnodelist` file:

```
Hp1.somecorp.com root # WebSphere database cluster
Hp2.somecorp.com root # WebSphere database cluster
```

If you want to allow non-root users to run **cmviewcl**, you should also add the non-root user IDs to this file.

- b. Define name resolution services:

By default, MC/ServiceGuard uses `/etc/resolv.conf` to obtain the addresses of the cluster nodes. In case DNS is not available, you should configure the `/etc/hosts` file and configure `/etc/nsswitch.conf` to search the `/etc/hosts` file when other lookup strategies are not working.

3. Set up and configure the shared disk array:

- a. Connect the shared disk array to both nodes.
- b. Create volume groups, logical volumes, and mirrors using **pvccreate**, **vgcreate**, **vgextend**, **lvcreate**, and **lvextend**.
- c. Create cluster lock disks.
- d. Distribute volume groups to the other node. You can distribute volume groups using either SAM or LVM commands.

Figure 12-5 on page 455 shows a two-node cluster. The disks are configured so that resources can be allocated to each node and each node may adopt the database application from the other node. Each database application has one disk volume group assigned to it and the logical volumes in that volume group are mirrored. Our arrangement eliminates single points of failure and makes either the disks or its mirrors available in the event that one of the buses fails.

4. Configure MC/ServiceGuard cluster for WebSphere databases:

- a. Using SAM, select **Cluster -> High Availability Cluster**.
- b. Choose **Cluster Configuration**.
- c. Select the **Actions** menu, and choose **Create cluster configuration**, then follow the instructions.

- d. Verify the cluster configuration using:
    - For DB2:
 

```
cmeckconf -k -v -C /etc/cmcluster/webspheredb2.config
```
    - For Oracle:
 

```
/etc/cmcluster/websphereoracle.config
```
  - e. Distribute the binary configuration file to the other node using either SAM or the command line.
  - f. Back up the volume group and cluster lock configuration data for possible replacement of disks later on.
5. Configure packages and their services:
- a. Install DB2 or Oracle in both machines and LDAP into the shared disk.
  - b. Create database instances into the shared LVG.
  - c. Use SAM to configure packages.
  - d. Customize the package control scripts for VG activation, service IPs, volume groups, service start, and service stop. Since the control scripts are very long, we give key sections of our sample scripts for DB2 and Oracle as follows:
    - For DB2, our sample service start script is:
 

```
function customer_defined_run_cmds
{
    su - db2inst4<<STARTDB
    db2start
    STARTDB
    test_return 51
}
```
    - Our sample DB2 service stop script is:
 

```
function customer_defined_halt_cmds
{
    su - db2inst4<<STOPDB
    db2 force applications all
    sleep 1
    db2stop
    STOPDB
    test_return 52
}
```

- For Oracle, our sample service start script is:

```
function customer_defined_run_cmds
{
    su - oracle<<STARTDB
    lsnrctl start
    export SIDS="APP ADMIN SESSION"
    for SID in $SIDS ; do
        export ORACLE_SID=$SID
        echo "connect internal\nstartup\nquit" | svrmgrl
    done
    STARTDB
    test_return 51
}
```

- Our sample Oracle service stop script is:

```
function customer_defined_halt_cmds
{
    su - oracle<<STOPDB
    export SIDS="APP ADMIN SESSION"
    for SID in $SIDS ; do
        export ORACLE_SID=$SID
        echo "connect internal\nshutdown\nquit" | svrmgrl
    done
    lsnrctl stop
    STOPDB
    test_return 52
}
```

- e. Distribute the package configuration using SAM.
6. Verify the cluster operation and configuration to ensure that:
  - Heartbeat networks are up and working normally
  - Networks are up and working normally
  - All nodes are up and working normally
  - All properties configured are correct
  - All services such as DB2, Oracle, LDAP are up and working normally
  - Logs should not have errors
7. Verify system failover from SAM by moving packages from one node to another.

## 12.2.4 Typical failover

Figure 12-5 on page 455 and Figure 12-6 on page 455 show a typical MC/ServiceGuard failover. During a typical database node failover, MC/ServiceGuard will:

- ▶ Stop database services and release resources in a failed node

- Unmount file systems
- Deactivate volume groups
- Activate volume groups in the standby node
- Mount file systems
- Assign package IP addresses to the LAN card on the standby node
- Execute database startup and get the needed resources

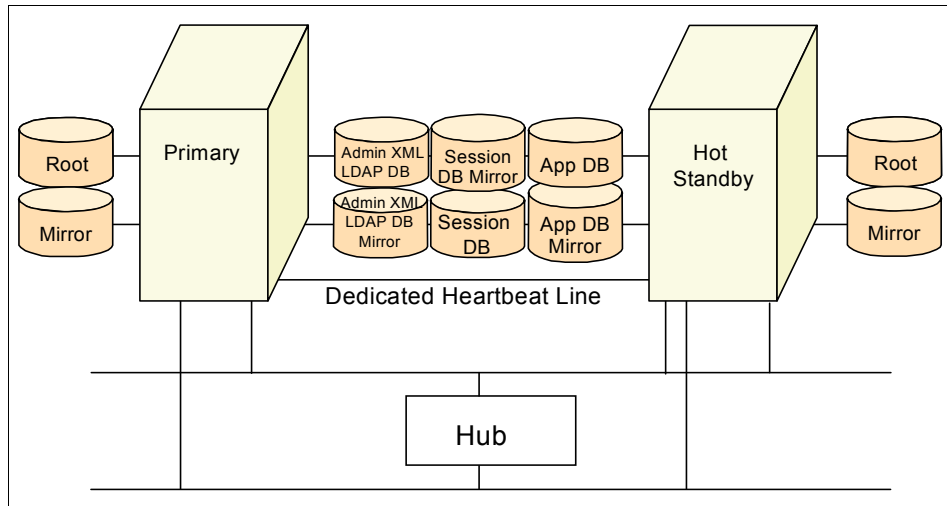


Figure 12-5 MC/ServiceGuard cluster configuration for WebSphere

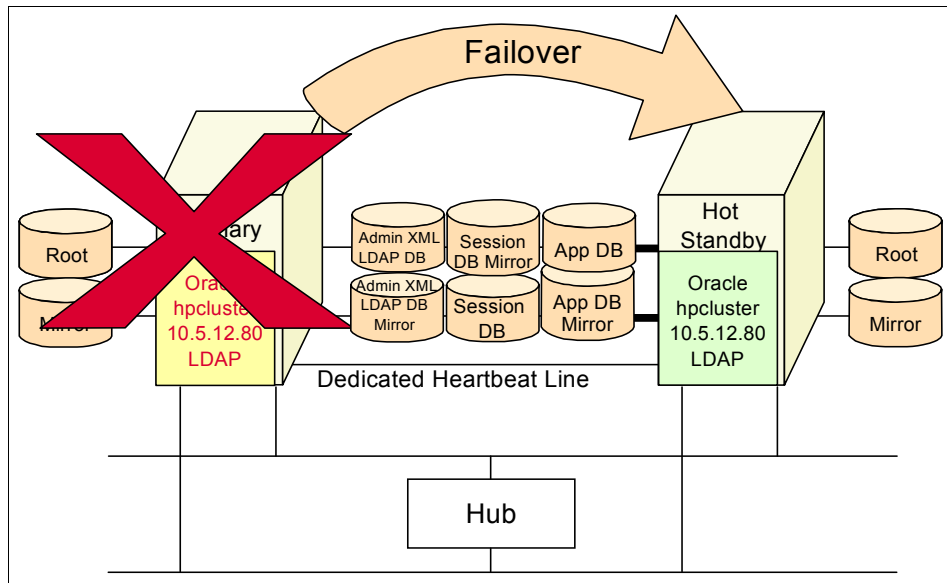


Figure 12-6 MC/ServiceGuard cluster after failover

## 12.2.5 Initiating various failures and tests

We performed both individual tests for each EJB kind, servlets, JSPs, applets and Java clients, as well as two-day stress testing of a WebSphere application (Trade3). We also initiated various failures.

For HA database nodes:

1. Kill the database processes to verify that automatic failover works:
  - a. Obtain PIDs for database processes (DB2, Oracle) using `ps -ef | grep <name>`.
  - b. Kill the database processes using `kill <PIDs>`.
  - c. View the database package status using `cmviewcl -v`.  
You should see the database package running on the specified adoptive node.
  - d. You can further verify the database processes in that node.
  - e. Your WebSphere server and WebSphere clients should function normally after the failover has finished (failover time).
  - f. Move the database package back to the primary node by using SAM.
2. Stop the database processes to verify that automatic failover works:
  - a. Stop the database servers using `db2stop` for DB2 or `shutdown` for Oracle.
  - b. View the database package status using `cmviewcl -v`.  
You should see the database package running on the specified adoptive node.
  - c. You can further verify the database processes in that node.
  - d. Your WebSphere server and WebSphere clients should function normally after the failover.
  - e. Move the database package back to the primary node by using SAM.
3. Turn off the power for the node to verify that automatic failover works:
  - a. Turn off the power for the primary node.
  - b. Observe the automatic database package failover using `cmviewcl -v`.  
The database package should be switched automatically to the backup node.
  - c. You can further verify the database processes in that node.
  - d. Your WebSphere server and WebSphere clients should function normally after the failover.



- e. Turn on the power for the primary node and observe that the primary node is rejoining the cluster by using `cmviewcl -v`.
  - f. Move the database package back to the primary node, using SAM.
4. Disconnect the network to the primary node to verify that automatic failover works:
  - a. Identify the primary and standby LAN cards using `lanscan` and `cmviewcl -v`.
  - b. Disconnect the LAN connection from the primary card.
  - c. Observe local or remote switching using `cmview -v`.
  - d. Your WebSphere server and WebSphere clients should function normally after the failover takes place.
  - e. Reconnect the LAN to the primary card of the primary node and check its status using `cmviewcl -v`.
  - f. Move the database package back to the primary node by using SAM.
5. Use SAM to further test whether packages can be moved from one node to another.

For the other parts in our WebSphere test topologies, we initiated failures by:

- ▶ Killing HTTP, application server, administrative server (Node Agent and Deployment Manager), and Load Balancer processes
- ▶ Stopping HTTP, application server, administrative server (Node Agent and Deployment Manager), and Load Balancer processes
- ▶ Disconnecting the network cable to each node
- ▶ Powering off each node, one at a time

We initiated various kinds of failures in our test topologies, and we counted the failed client requests versus the total client requests. The failure rate was very small (less than 1%), depending on the frequency of initiation of various failures, MC/ServiceGuard detection and recovery time, and network characteristics. We found that WebSphere process failures were instantaneously recovered through the WebSphere WLM mechanism and contributed very little to failed requests. However, database recovery took minutes to complete and contributed the most to the failed requests. Therefore, in the next section, we discuss how to tune the clustering parameters.

## 12.2.6 Tuning heartbeat and cluster parameters

Sending and receiving heartbeat messages among the nodes in the cluster is key for the cluster manager. If a cluster node does not receive heartbeat messages from the other node within the prescribed time, a cluster reformation is initiated. At the end of reformation, if a new set of nodes form a cluster, that information is passed to the package coordinator. Packages running on nodes that are no longer in the new cluster are transferred to their adoptive nodes.

A package is a collection of services, disk volumes and IP addresses managed by MC/ServiceGuard to ensure that they are available. Packages are units of failover by which MC/ServiceGuard starts and halts configured applications. The package coordinator decides when and where to run, halt or move packages. The package manager executes the user-defined control script to run and halt packages and package services, and react to changes in the monitored resources. A package starts up on the primary node when the cluster starts. In case of a service, network, or resource failure, a package failover takes place when the package coordinator initiates the start of a package on a new node. A package failover involves both halting the existing package and starting the new instance of the package. A package switch involves moving packages and their IP addresses to a new system. With package failovers, TCP connections are lost. TCP applications must reconnect to regain connectivity; this is not handled automatically.

The network manager detects and recovers from network card and cable failures so that network services remain highly available to clients. There are stationary IP addresses for each active network interface in each node. For example, in our case, we used 10.4.12.100, 10.5.12.200, etc. In order to perform transparent failover, we need to assign a relocatable IP address to database services so that clients will use the same IP address to connect to database servers after database services failover to another node. Our relocatable IP address for database servers is 10.5.12.80 or hpcluster.itso.ibm.com. The relocatable IP address is also known as a *floating IP address* or *package IP address*. When the database package starts up, the **cmmodnet** command in the package control script assigns this relocatable IP address (hpcluster.itso.ibm.com) to the primary LAN interface card in the primary node. Within the same node, both stationary and relocatable IP addresses will switch to a standby interface in the event of a LAN card failure. Only a relocatable IP address can be taken over by an adoptive node if control of the package is transferred. Therefore, WebSphere can access the database server via its relocatable IP address without knowing on which node the database server is running.

At regular intervals, MC/ServiceGuard polls all the network interface cards specified in the cluster configuration file. The polling interface sends LAN packets to all other interfaces in the node that are on the same bridged net and receives

packets back from them. Once a network failure is detected, if there is a backup LAN card in the node, a local network switch is initiated and TCP/IP connections are not lost (except for IEEE 802.3 that does not have the rearp function). Otherwise, remote switch is initiated, TCP connections are lost, and database applications (WebSphere) must reconnect to regain connectivity. A remote switch involves moving packages and their associated IP address to a new system.

MC/ServiceGuard, together with other software, also monitors other resources and takes action when events occur. These resources are monitored by MC/ServiceGuard as a package dependency.

There are several parameters you can tune for performance and reliability:

- ▶ The heartbeat interval is the normal interval between the transmission of heartbeat messages from one node to the other in the cluster. The default value is 1 second, but you can set it to up to 30 seconds. We tested several different values, then used 2 seconds.
- ▶ The node timeout is the time after which a node may decide that the other node has become unavailable and initiate cluster reformation. The default value is 2 seconds, with a minimum of 2 x (heartbeat interval) and a maximum of 60 seconds. Small values of node timeout and heartbeat interval may increase the potential for spurious cluster reformation due to momentary system hangs or network load spikes. We tested several different values, then set 2 seconds of heartbeat interval and 4 seconds of node timeout.
- ▶ The network polling interval is the frequency at which the networks configured for MC/ServiceGuard are checked. The default value is 2 seconds. Changing this value can affect how quickly a network is detected, ranging from 1 to 30 seconds.
- ▶ Switching an IP address from a failed LAN card to a standby LAN card on the same physical subnet may take place if automatic switching is set to Enabled in SAM. You can define failover behavior and whether it will fall back automatically as soon as the primary node is available.
- ▶ The resource polling interval is the frequency of monitoring a configured package resource. The default value is 60 seconds, with a minimum value of 1 second.

## 12.3 WebSphere with Sun Cluster

In this section, we look at integrating the WebSphere system with Sun Cluster. The topics discussed are:

- ▶ Introduction and considerations
- ▶ Setup and configuration

### 12.3.1 Introduction and considerations

Both Sun Cluster and VERITAS Cluster use agents to greatly simplify cluster configuration. Each type of resource supported in a cluster is associated with an agent. An agent is an installed program designed to control a particular resource type. For Sun Cluster or VERITAS Cluster to bring an Oracle resource online, it does not need to understand Oracle; it simply passes the online command to the Oracle agent. The Oracle agent knows to call the server manager and issue the appropriate command. Both Sun Cluster and VERITAS cluster have Oracle, DB2, Informix®, and SyBase agents. Many other enterprise agents are also available. Some of these agents come with the base cluster software, and some require an additional purchase. Sun Cluster can also use the VERITAS Volume Manager.

Sun Cluster supports up to eight nodes while VERITAS supports up to 32 nodes. Sun Cluster is integrated with OPS and OPFS, but VERITAS Cluster does not support OPS and OPFS.

The Sun Cluster, or SunPlex system, is an integrated hardware and software solution that is used to create highly available and scalable services. The SunPlex system extends the Solaris operating environment into a cluster operating system. The commonly used versions are Sun Cluster 2.2 and Sun Cluster 3.0. There are some differences between the versions. For example, Sun Cluster 2.2 requires a terminal concentrator for failure fencing, but later products do not depend on the terminal concentrator.

The Sun Cluster we configured for WebSphere is shown in Figure 12-7 on page 461. It can:

- ▶ Reduce or eliminate system downtime because of software or hardware failures.
- ▶ Ensure availability of data and applications to end users, regardless of the kind of failure that would normally take down a single-server system.
- ▶ Increase application throughput by enabling services to scale to additional processors by adding nodes to the cluster.
- ▶ Provide enhanced availability of the system by enabling you to perform maintenance without shutting down the entire cluster.

Sun Cluster configurations, as shown in Figure 12-7 on page 461, tolerate the following types of single-point failures:

- ▶ Server operating environment failure because of a crash or a panic
- ▶ Data service failure
- ▶ Server hardware failure
- ▶ Network interface failure
- ▶ Disk media failure

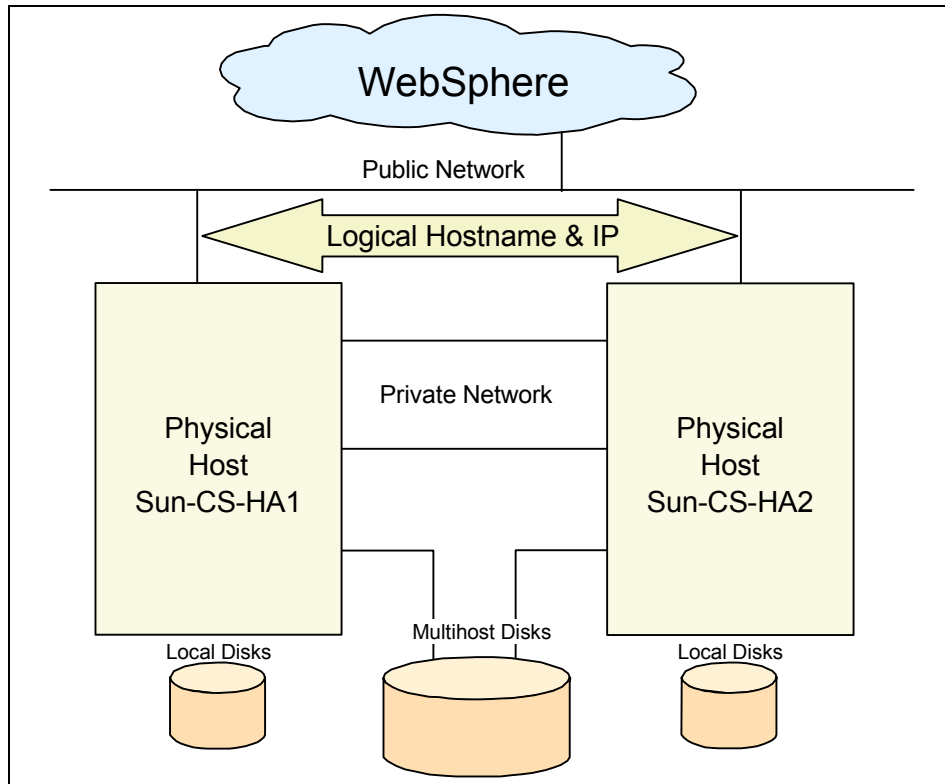


Figure 12-7 Sun Cluster configuration for WebSphere

### 12.3.2 Setup and configuration

The SunPlex system achieves high availability through a combination of hardware and software. The redundant cluster interconnects storage and public networks to protect against single points of failure. The cluster software continuously monitors the health of member nodes and prevents failing nodes from participating in the cluster, protecting against data corruption. Also, the cluster monitors services and their dependent system resources, and fails over or restarts services in case of failures. There are several steps to configure a Sun Cluster for WebSphere:

1. Connect and configure the shared disk subsystem:

Database data and log files are stored in a mirrored or RAID shared disk array that can be connected physically to two nodes (multihost disk). When one node fails, the other node can access the same data and log files.

You can configure the shared disk using either Sun Cluster Solstice DiskSuite or VERITAS Cluster Volume Manager.

2. Install and configure Sun Cluster data service software:

Install the Sun Cluster software. Sun Cluster will use the **su** command when it starts or stops the database server. In order to avoid the impact of failed network information name service during failure of the public network on the cluster node, modify the `/etc/nsswitch.conf` file on each node where the logical host will run so that group lookups are directed to files first.

3. Install and configure DB2 and Oracle:

You can select to install DB2 or Oracle binaries either on the local disks of each node or on the multihost shared disks. Placing database binaries on the multihost disk eases administration, since there is only one copy to administer. However, it sacrifices redundancy and therefore availability in some failures. Placing database binaries on the local disk of the physical host increases redundancy and therefore availability. However, placing the binaries on the local disk increases the administrative overhead, since you must manage multiple copies of the files. If you select to install on the local disks, the version of the database must be the same.

When first installing DB2 or Oracle, select the **Install only** option. This is necessary because database initialization and configuration files must be modified to reflect the logical hosts as the location for the database. Install DB2 or Oracle as usual (see the corresponding DB2 or Oracle product manuals).

4. Make DB2 and Oracle database cluster-ready for failover:

Start the Sun Cluster and take ownership of the disk group using:

```
scadmin startcluster
```

Make sure that the multihost disks have been configured properly. Create the DB2 instance or Oracle database. After that, you must make it cluster-ready. Register the database service using:

```
hareg -s -r Oracle -h was1h
```

Activate the database service using:

```
hareg -y Oracle
```

And finally set Sun Cluster to monitor the database instance and bring the database into service.

WebSphere must use the logical host name (not the physical host name) to access the database. When the database server fails from the primary host to the secondary host, you do not need to change the WebSphere database connection string, since the logical host name is the same before and after a failover.

5. Verify the cluster configuration:

You can perform a verification using **haswitch**. You can also check the status using **hastat**.

## 12.4 WebSphere with VERITAS Cluster

In this section, we look at integrating the WebSphere system with VERITAS Cluster. The topics discussed are:

- ▶ Introduction and considerations
- ▶ Setup and configuration

### 12.4.1 Introduction and considerations

VERITAS Cluster Server (VCS), similar to Sun Cluster, provides an integrated cluster framework for controlling and monitoring services for high availability. The unit of failover in VCS is a service group. A service group can be used to define a collection of resources such as disks, IP addresses, and processes.

Configuring VERITAS Cluster Server is similar to configuring the Sun Cluster as discussed above. A typical configuration, shown in Figure 12-8 on page 464, includes two nodes interconnected by private networks and a shared disk.

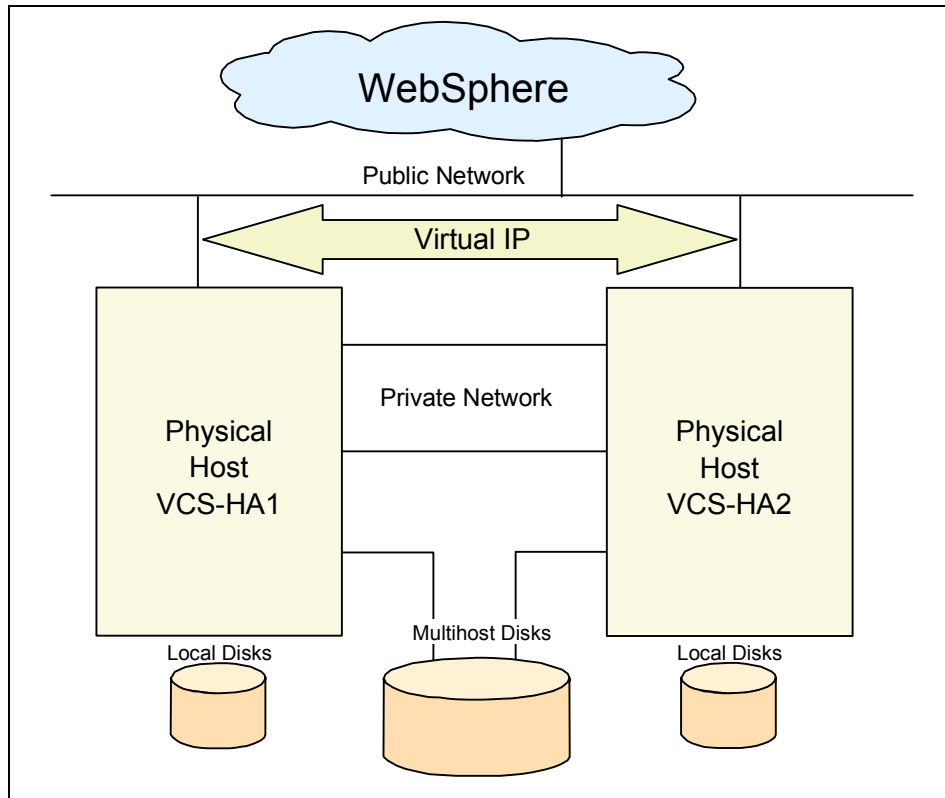


Figure 12-8 VERITAS Cluster Server configuration for WebSphere

## 12.4.2 Setup and configuration

We set up and configured our test environment as follows:

1. Set up the network and shared disk:

As discussed above, set up the public networks and private networks. Physically connect both nodes to a shared disk and configure the logical volume (VxVM) group that can be accessed by both nodes, although not concurrently.

2. Configure a cluster:

Install VCS according to the manual. Create a cluster and configure the cluster with the public and private networks.



3. Create the database instance for WebSphere:  
Install Oracle on both nodes. Create a database instance home directory in the disk group owned by the service group, create the database owner user and specify the home directory just created. Create the Oracle database for the persistent session and application databases.
4. Configure a service group for the WebSphere databases:  
Configure the database instance under VCS control by configuring the database agent and modifying the cluster configuration. Add resources by using VCS GUI or editing the main.cf and type.cf files. Verify and enable the configuration using **hacf -verify**.
5. Test the cluster:  
Verify the cluster by taking the resource group offline and online. Test the cluster by taking the resource group online and by initiating some faults such as killing the process. The monitor should correctly detect failures and failover. Test the WebSphere system as discussed in the previous sections.

## 12.5 WebSphere with Microsoft Cluster Service

In this section, we look at integrating the WebSphere system with Microsoft Cluster Service. The topics discussed are:

- ▶ Introduction and considerations
- ▶ Setup and configuration

### 12.5.1 Introduction and considerations

The Microsoft Cluster Service (MSCS) allows you to configure two-node high availability clusters on standard PC server hardware. MSCS server clusters provide redundancy and fault resilience for the WebSphere and application data services. The Microsoft Windows NT® Server Enterprise Edition and the Windows 2000 Advanced Server include MSCS.

MSCS uses a share-nothing clustering architecture, which has no concurrently shared resources. It works by transferring the ownership of resources from one node to another automatically to keep data services available in case of failures. You can also manually switch from one node to another using operator commands. MSCS provides the ability to group resources and express dependencies between them, control process starting and stopping in a logical sequence, and move them from one node to another. It is also possible to configure a failback policy that controls whether an application moves back to its former node when that node becomes available again.

WebSphere is configured to use virtual IP addresses that are under MSCS cluster control, as shown in Figure 12-9. When database services (DB2, Oracle, MS SQL Server) move from one clustered node to the other, it takes its virtual IP address with it. The virtual IP address is different from the stationary physical IP address. The virtual IP address make database failover transparent to WebSphere and other application clients.

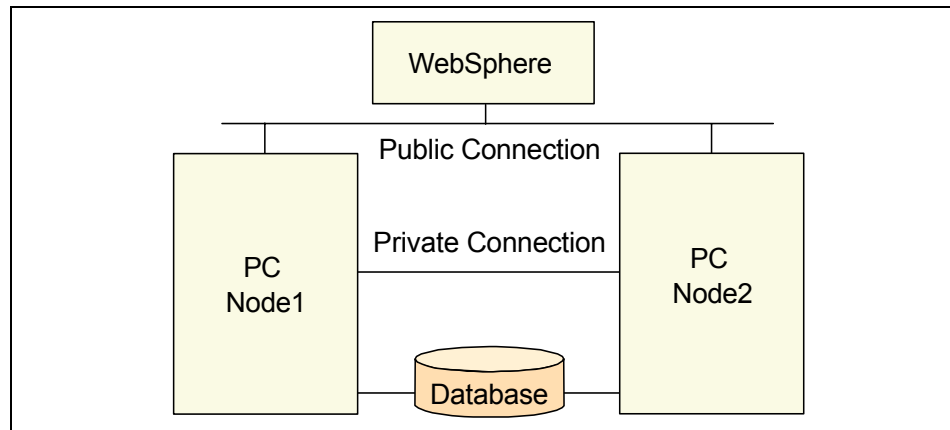


Figure 12-9 Microsoft Cluster Service configuration for WebSphere

## 12.5.2 Setup and configuration

It is easy to configure the MSCS cluster using the Microsoft graphical setup wizard. Cluster service setup requires less than 10 mouse clicks on the first cluster node, and less than four on the second. There are many documents and books that discuss MSCS configurations. Microsoft also provides a detailed step-by-step configuration guide at:

<http://microsoft.com/windows2000/techinfo/planning/server/clustersteps.asp>

The minimum hardware requirements for a MSCS cluster are two PC servers, a shared disk, a private network (Ethernet) and a public network (Ethernet or token-ring). As an option, you can add additional networks and network adapters to eliminate network single points of failures. The operating system can be either Windows NT or Windows 2000. You must use identical versions of the operating system on both PC server nodes in a cluster.

The Microsoft graphical setup wizard makes the installation and configuration of MSCS very easy. It is highly automated. During the installation process, some nodes will be shut down and some nodes will be rebooted to protect data in your shared disk. When multiple nodes try to simultaneously write to the same disk that is not yet protected by the cluster software, the data on your shared disk may

be lost or corrupted. Therefore, you need to follow the power sequencing for cluster installation as described in the Microsoft step-by-step guide. There are several steps to configure a Microsoft Cluster Service:

1. Set up the base operating system, disks, and networks:

You need to install the base operating system (for example, Windows 2000 Advanced Server or Windows 2000 Datacenter Server). Each node should have two network adapters, for the public and private networks. You need to have one IP address for the cluster with which WebSphere will connect, and four additional IP addresses for the public and private networks. All shared disks must be configured as basic (not dynamic) and formatted as NTFS. If you use SCSI devices, you need to assign unique SCSI identification numbers.

2. Install MSCS on one node, create the cluster in progress:

Use the Cluster Service Configuration wizard. You must shut down other nodes and power on the shared disk before you do this.

3. Install MSCS on the other node, join the cluster:

After you finish the first node and shared disk installation, start the Cluster Service Configuration wizard on the second node. Installing the cluster service on the second node requires less time, since some settings are available from the first node. You must make sure that the cluster service is running on the first node and the shared disks are powered up. Select to join the cluster you created on the first node. You can check the cluster by selecting **Start -> Programs -> Administrative Tools -> Cluster Administrator**.

4. Database (DB2, Oracle, SQL Server) and instance creation:

Install the database software on each node as usual (see the manual for each database product). For SQL Server, just run the Cluster Failover wizard. For others, you either can use a utility such as db2mscs, or configure the group manually with the Microsoft wizard.

5. Test the cluster configuration:

Connect WebSphere to the database with the cluster IP address. The first verification test would be to try to move the database group from one node to another using **Start -> Programs -> Administrative Tools -> Cluster Administrator**. Click the group you selected, and choose **Move**. Then you can try to stop services, disconnect cables, and shut down nodes.

## 12.6 Programming clients for transparent failover

For a Web client, requests are directed to one of many Web servers (through a sprayer such as the Load Balancer that is part of the WebSphere Network Deployment Edge Components), then redirected to one of many WebSphere Web containers (through plug-in WLM). Requested servlets may use EJBs or data sources. For a servlet, you may elect to bootstrap to one of many application servers or Node Agents or Deployment Manager (where the naming service is hosted, WebSphere V5.1 hosts a naming service in all of its server processes), then create and/or find a bean or a data source.

Similarly, for a Java client, you may elect to bootstrap to one of many application servers or Node Agents or Deployment Manager, then create and/or find a bean or a data source.

WebSphere groups `SQLExceptions` that may cause stale connections in a new exception called `StaleConnectionException` that extends `SQLException` (`com.ibm.websphere.ce.cm.StaleConnectionException`).

If you use the optimistic concurrency, you can also catch the `OptimisticUpdateFailureException` or `TransactionRolledBackException` and retry.

### 12.6.1 Bootstrap to multiple hosts

If clients use only a single bootstrap server, they cannot get an `InitialContext` if the WebSphere Application Server or Node Agent or Deployment Manager fails. Using multiple bootstrap servers will improve availability, and WebSphere V5.1 provides an easy way to use multiple bootstrap servers in its `corbaloc` or `corbaname`. If one of many bootstrap servers is available, clients can still get an `InitialContext`. Once an `InitialContext` has been obtained, the application will be workload managed.

However, in WebSphere V5.1, the Node Agent is not workload managed, and you need to clean the naming cache and to retry explicitly if you want to use multiple Node Agents as bootstrap servers. We suggest that you use clustered application server members for multiple bootstrap servers, since they are workload managed. Unless you use the high availability solution of the Deployment Manager (as described in Chapter 10, “Deployment Manager and Node Agent high availability” on page 389), you should not use the Deployment Manager as the bootstrap server alone.

## 12.6.2 Catch StaleConnectionException

WebSphere groups a subset of SQLExceptions that may indicate invalid connections into a new exception, StaleConnectionException. We can catch StaleConnectionException and code a retry block in clients.

Clients can be servlets, EJBs, Java programs, etc. In the servlets, an auto commit, local transaction, or global transaction is usually used. In Java clients, an auto commit or local transaction is usually used. In the EJBs, a global transaction is usually used. We can distinguish direct JDBC calls and indirect JDBC calls. For indirect JDBC calls, we also can distinguish rethrowable, new retryable exceptions, or those that are wrapped in another exception.

### Direct JDBC calls, auto commit

This is for single statement transactions. A segment of sample code is shown in Example 12-1.

*Example 12-1 StaleConnectionException with direct JDBC calls and auto commit*

---

```
...
    javax.sql.DataSource ds=null;
    java.sql.Connection conn=null;
    java.sql.Statement stmt1=null;
    java.sql.ResultSet rs=null;
    int maxRetries=5; // set maximum number of retries
    int noRetries=0;
    int bal=0;
    boolean retry=false;

    do {
        try { //already lookup datasource object ds, see above section
            conn=ds.getConnection();
            conn.setAutoCommit(true);
            stmt1=conn.createStatement();
            stmt1.executeUpdate(
                "UPDATE SAVINGS set balance=500 where accountID=10001");
        } catch (com.ibm.websphere.ce.cm.StaleConnectionException staleex) {
            if (noRetries <= maxRetries) {
                // retry upto maxRetries times
                try {
                    Thread.sleep(1000 + 20000*noRetries);
                    // with increasing retry interval
                    // preserve cpu cycle
                } catch (InterruptedException iex) {
                    System.out.println("Sleep Interrupted" + iex.getMessage());
                }
                noRetries++;
                retry=true;
            }
        }
    }
}
```

```

        } else {
            retry=false;
            // don't retry after maxRetries times
        }
    } catch (java.sql.SQLException sqlex) {
        //other sql exception, don't retry
        retry=false;
    } catch ( Exception ex) {
        // generic exception, don't retry
        retry=false;
    } finally {
        try {
            if (rs!=null) rs.close();
            if (stmt1!=null) stmt1.close();
            if (conn!=null) conn.close();
        } catch (Exception ex) {}
    }
} while (retry);
...

```

---

There is another WebSphere connection pool exception called `ConnectionWaitTimeoutException` (`com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException`) that extends `SQLException`. This exception indicates that the application has waited for the connection timeout (`CONN_TIMEOUT`, by default 180) number of seconds and has not been returned a connection. We do not suggest that you retry when this exception occurs. Instead, you can increase your connection pool size and/or `CONN_TIMEOUT` value.

## Direct JDBC calls, local transaction

This can group several statements in a unit of work with `setAutoCommit` set to `false`, as shown in Example 12-2.

*Example 12-2 StaleConnectionException with direct JDBC calls and local transaction*

---

```

...
    javax.sql.DataSource ds=null;
    java.sql.Connection conn=null;
    java.sql.Statement stmt1=null;
    java.sql.ResultSet rs=null;
    int maxRetries=5; // set maximum number of retries
    int noRetries=0;
    int bal=0;
    boolean retry=false;

    do {
        try { //already lookup datasource object ds, see above section
            conn=ds.getConnection();

```

```

        conn.setAutoCommit(false);
        stmt1=conn.createStatement();
        rs=stmt1.executeQuery(sqlstring1);
        bal=rs.getInt(3);
        stmt1.executeUpdate(sqlstring2);
        conn.commit();
    } catch ( com.ibm.websphere.ce.cm.StaleConnectionException staleex) {
        //rollback
        try{
            conn.rollback();
            if (noRetries <= maxRetries) {
                // retry upto maxRetries times
                try {
                    Thread.sleep(1000 + 20000*noRetries);
                    // with increasing retry interval
                    // preserve cpu cycle
                } catch (InterruptedException iex) {
                    System.out.println(
                        "Sleep Interrupted" + iex.getMessage());
                }
                noRetries++;
                retry=true;
            } else {
                retry=false;
                // don't retry after maxRetries times
            }
        } catch (Exception ex) {
            retry=false;
            //rollback failed, don't retry
        }
    } catch (java.sql.SQLException sqlex) {
        //other sql exception, don't retry
        retry=false;
    } catch ( Exception ex) {
        // generic exception, don't retry
        retry=false;
    } finally {
        try {
            if (rs!=null) rs.close();
            if (stmt1!=null) stmt1.close();
            if (conn!=null) conn.close();
        } catch (Exception ex) {}
    }
} while (retry);
...

```

---

## Direct JDBC calls, global transaction

This, also, can group several statements in a unit of work, as shown in Example 12-3. A `javax.transaction.UserTransaction` object can be retrieved from JNDI naming or from an `EJBContext` object (for example, `getSessionContext().getUserTransaction()` in session beans).

*Example 12-3 StaleConnectionException with direct JDBC calls and global transaction*

---

```
public class RetryBMTSessionEJBBean implements SessionBean {
    private javax.ejb.SessionContext mySessionCtx = null;
    private final static long serialVersionUID = 3206093459760846163L;

    public void updateAccount(String id, float amount) {
        UserTransaction ut=getSessionContext().getUserTransaction();
        Statement stmt=null;
        ResultSet rs=null;
        Connection conn=null;
        int maxRetries=5;
        int retryNum=0;
        boolean retry=false;
        do {
            try {
                ut.begin();
                conn=ds.getConnection();
                stmt=conn.createStatement();
                stmt.executeUpdate(sqlstring1);
                stmt.executeUpdate(sqlstring2);
                ut.commit();
            } catch (com.ibm.websphere.ce.cm.StaleConnectionException scex){
                ut.rollback();
                if (retryNum < maxRetries) {
                    retry=true;
                    retryNum++;
                }
                retry=false;
            } catch (SQLException sqllex) {
            } catch (Exception ex) {}
        } while (retry);
    }
}
```

---

## Indirect JDBC calls, rethrowable

Another exception such as `RetryItException` can be rethrown so that you can catch it and retry it in your clients. A segment of code sample is shown in Example 12-4.

*Example 12-4 StaleConnectionException with indirect JDBC calls and rethrowable*

---

```
public class RetryEJBBean implements EntityBean {
    private javax.ejb.EntityContext entityContext = null;
```



```

        private final static long serialVersionUID = 3206093459760846163L;

    public void update(String name) throws RetryItException, RometeException
    {
        try {
            conn=ds.getConnection();
            conn.setAutoCommit(false);
            stmt=conn.createStatement();
            stmt.executeUpdate(sqlstring1);
        } catch (com.ibm.ce.cm.StaleConnectionException sce) {
            getEntityContext().setRollbackOnly();
            throw new RetryItException(sce.getMessage());
        } catch (SQLException sqllex) {
            ...
        }
        ...
    }

    public class RetryEJBClient {

    public static void main(java.lang.String[] args) {
        //this also can be a part of session bean
        boolean retry=false;
        int maxRetries=5;
        int retryNum=0;
        do {
            try {
                RetryEJB thisEJB=retryEJBHome.findByPrimaryKey(aKey);
                thisEJB.update(newName);
            } catch (RetryItException rex) {
                if (retryNum < maxRetries) {
                    retry=true;
                    retryNum++;
                } else {
                    retry=false;
                }
            }
            try {
                Thread.sleep (1000);
            } catch (InterruptedException iex) {}
        } catch (Exception ex) {}
        } while (retry);
    }
}

```

---

## Indirect JDBC calls, wrapped

You can inspect the details of `RemoteException` to check whether it originates from a retryable exception. When a `RemoteException` is thrown to wrap another exception, the original exception is usually retained. All `RemoteExceptions` have an attribute `detail`, which is of type `java.lang.Throwable`. With this detail, we can trace back to the original exception to see whether its root cause is retryable. However, the detail is lost if a `RemoteException` flows from one JVM to another. A segment of code sample is shown in Example 12-5.

*Example 12-5 Wrapping `StaleConnectionException`*

---

```
public boolean retryableException(RemoteException rx) {
    RemoteException rex=rx;
    Throwable th=null;
    while (true) {
        th=rex.detail;
        try {
            rex= (RemoteException) th;
        } catch (ClassCastException cex) {
            return (
                th instanceof com.ibm.websphere.ce.cm.StaleConnectionException);
        }
    }
}
```

---

Then you can use the sample shown in Example 12-6.

*Example 12-6 `StaleConnectionException` wrapped with indirect JDBC calls*

---

```
public class WrappedRetrySessionEJBBean implements SessionBean {
    private javax.ejb.SessionContext mySessionCtx = null;
    private final static long serialVersionUID = 3206093459760846163L;

    public void updateAccount() {
        javax.transaction.UserTransaction ut=
            getSessionContext().getUserTransaction();
        boolean retry=false;
        int maxRetries=5;
        int retryNum=0;
        do {
            try {
                ut.begin();
                aBean.update(id, deposit);
                //must catch SCE and rethrow it in remote exception
                ut.commit();
            } catch (RemoteException rex) {
                try {
                    ut.rollback();
                } catch (Exception ex) {}
            }
        }
    }
}
```

```

        retry=false;
        if (retryableException(rex) && retryNum<maxRetries) {
            retry=true;
            retryNum++;
        }
    } catch (Exception ex) {}
} while (retry);
}

```

---

#### *Example 12-7 RemoteException*

---

```

...
boolean retry=false;
int retries=maxRetries; // set the maximum number of retries
do {
    try {
        aBean.update(xxx);
        break;
    } catch (RemoteException rex) {
        retry=false;
        if (retryableException(rex) && retries > 0) {
            retry=true;
            retries--;
        }
        try {
            Thread.sleep(1000); //sleep a while
        } catch (InterruptedException iex) {}
    }
} while (retry);
...

```

---

## 12.7 WebSphere and IP-based database failover

This section examines WebSphere behaviors during and after database failovers for IP-based highly available data management systems.

### 12.7.1 Administrative servers and administrative actions

IBM WebSphere Application Server Network Deployment V5.1 does not use a central repository to store configuration information as WebSphere V4 does. When the database is down, WebSphere V5 administrative server processes are unaffected and still run as usual; all administrative actions still function. We discussed WebSphere V5.1 master repository and Deployment Manager high availability in Chapter 10, “Deployment Manager and Node Agent high availability” on page 389.

## 12.7.2 Application servers

IBM WebSphere Application Server Network Deployment V5.1 does not rely on a database for its administrative repository. However, Entity EJBs rely on a database and MDBs rely on JMS providers. If the database fails, client requests to these entities will also fail. After the database is recovered, client requests will succeed after no more than one retry.

## 12.7.3 Persistent session

You have the options to put the session data in memory or to store it in a database. Thus, you can use the WebSphere V5 Data Replication Service (memory-to-memory replication) or database as mechanisms for session failover. If you choose to store your session data into a database, a highly available session database is achieved through clustering software or parallel database. During a failover, the session data will be temporally unavailable and any request that requires a persistent session will fail. After a failover, the session data will become available again and any request will succeed after the first retry.

## 12.7.4 Java/C++ applications and application re-connecting

During the failover, any in-flight application that uses database access will fail. After the failover, any application will succeed on retry. For the Java applications, you can catch the `StaleConnectionException` in your database try block and retry your connection a limited number of times, with the sample code discussed in “Catch `StaleConnectionException`” on page 469.

We have three comments regarding this coding practice:

- ▶ This coding practice is not required for the HA solution to work; it is only an option. We tested both situations with and without this coding practice.
- ▶ A typical database failover takes approximately one to five minutes. If you code this try block in your application with an unlimited number of retries, then your application will hang there for one to five minutes during a failover. There is not much to gain by trying hundreds of times during a failover; it may impact the performance of other WebSphere and application processes that do not need to access the database. We suggest that you either do nothing in your application code, or code a limited number of retries (no more than five times with a controlled retry interval). We tested all of these scenarios and we suggest that you consider the trade-off for each option before you develop your application.
- ▶ You need to pay attention to transaction ACID properties (atomicity, consistency, isolation, and durability) when coding retry.

## 12.7.5 Enterprise beans

Entity EJBs store data in databases, and session beans may be coded directly with JDBC to access the database or work with Entity EJBs. EJBs that use database access will fail during a failover and succeed after a failover with no more than one retry. For CMP EJB, when a part of data access is not controlled by WebSphere containers and BMP EJB data access, you have the option to code a retry block in your code as described 12.7.4, “Java/C++ applications and application re-connecting” on page 476. If you code the retry block in your EJBs, clients will hang during a failover. For CMP EJBs, restarting application servers will clean the connection pool after a failover.

## 12.7.6 Web-based clients, applets, servlets, and JSPs

If these clients do not need database access, they run as usual, without interruption. If database access is needed, requests will fail until the failover has finished. In addition to direct database access, these clients may also use persistent HTTP sessions. Once the database failover has finished, with no more than one retry, these clients should run as usual.

Web-based clients are easy to retry on the client side (just go back in the browser and submit the request again). Therefore, coding a retry block is not as important as in applications. But you can choose to code the retry block in your servlets and applets.

## 12.7.7 Naming service and security service

WebSphere V5 does not use a database to store its name bindings. If a database is used for the security repository, users cannot log in during a failover, but will succeed after a failover.

In order to minimize lookup failures, you can code your applications as follows:

```
if (obj==null)
    ctx.lookup(obj);
```

Since the system has run for a while before a failure, the object you are looking up may already exist, so you do not need to look it up.

## 12.7.8 Workload management

No matter which cluster member is used in the WebSphere WLM environment, during a database failover all requests to access data will fail. After a failover, any request should succeed after no more than one retry. The workload-managed WebSphere Application Servers do prevent some node or process failures, as

tested in our topologies, but not database network, database node and database process failures. Availability has two distinct aspects: process availability and data availability. The WebSphere WLM mechanism protects the system from process failures. We still need HA data management for WebSphere data availability.

## **12.8 WebSphere with Oracle Parallel Server (OPS) and Real Application Cluster (RAC)**

In this section, we look at integrating your WebSphere system with Oracle Parallel Server and Real Application Cluster. The topics discussed are:

- ▶ Introduction and considerations
- ▶ Setup and configuration

### **12.8.1 Introduction and considerations**

Oracle Parallel Server (OPS) and Real Application Cluster (RAC) offer superior scalability through the distribution of workload across nodes, and high availability through multiple nodes accessing the database. If one node fails, the database is still accessible through the surviving nodes.

Approaches discussed above use the IP-failover mechanism and use clustering software to manage the resource group. The failover time is usually in the range of one to five minutes. Some parallel database servers, such as Oracle Parallel Server (OPS) and Real Application Cluster (RAC) not only provide greater scalability, but also facilitate faster failover, since the instance is pre-existent in the backup system. Although OPS is not registered with or managed by the cluster resource group manager, it still relies on a clustering software layer to query cluster information. Therefore, platform-specific clustering software such as HACMP, MC/ServiceGuard, or Sun Cluster, is required for OPS/RAC to function in a clustered environment. The OPS/RAC system consists of an operating system-dependent layer (platform-specific cluster software), Oracle 8i/9i Enterprise Edition, Oracle Parallel Server Option, and Oracle Parallel Server Management. Inside OPS, there are Integrated Distributed Lock Management (DLM), Parallel Cache Management (PCM), and Oracle Parallel Query.

You can configure OPS/RAC to use the shared-disk architecture of the cluster software. In this configuration, a single database is shared among multiple instances of OPS/RAC that access the database concurrently. The Oracle Distributed Lock Manager controls conflicting access to the same data. If a process or a node crashes, the DLM is reconfigured to recover from the failure. If a node failure occurs in an OPS/RAC environment, you can configure Oracle

clients to reconnect to the surviving server without the use of the IP failover, as detailed in 12.1, “WebSphere with IBM HACMP” on page 436 through 12.6, “Programming clients for transparent failover” on page 468. The multiple Oracle instances cooperate to provide access to the same shared database. The Oracle clients can use any of the instances to access the database. Thus, if one or more instances have failed, clients can connect to a surviving instance and continue to access the database.

## **12.8.2 Setup and configuration**

Setup and configuration of Oracle Parallel Server/Real Application Cluster consist of the following steps:

1. Install and configure the shared disk subsystem
2. Create raw devices
3. Install and configure the operating system-dependent layer
4. Install and configure OPS/RAC
5. Configure the OPS/RAC net service for failover
6. Configure WebSphere servers

You can configure OPS/RAC into a failover mode, as shown in Figure 12-10 on page 480, or a failover with load balance, as shown in Figure 12-11 on page 481.

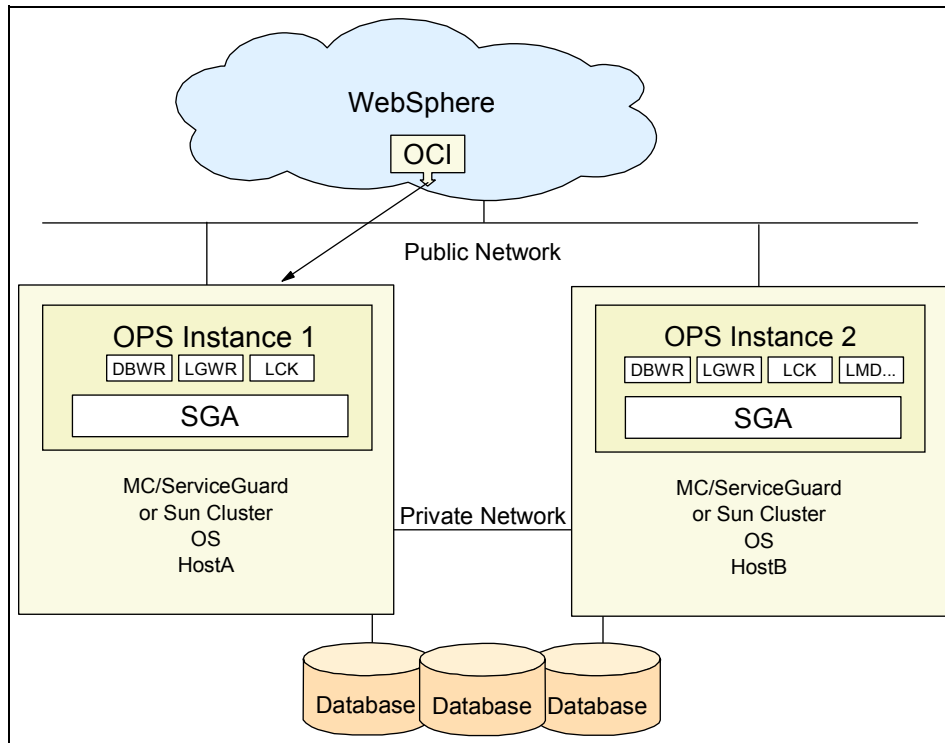


Figure 12-10 OPS/RAC configuration without load balancing



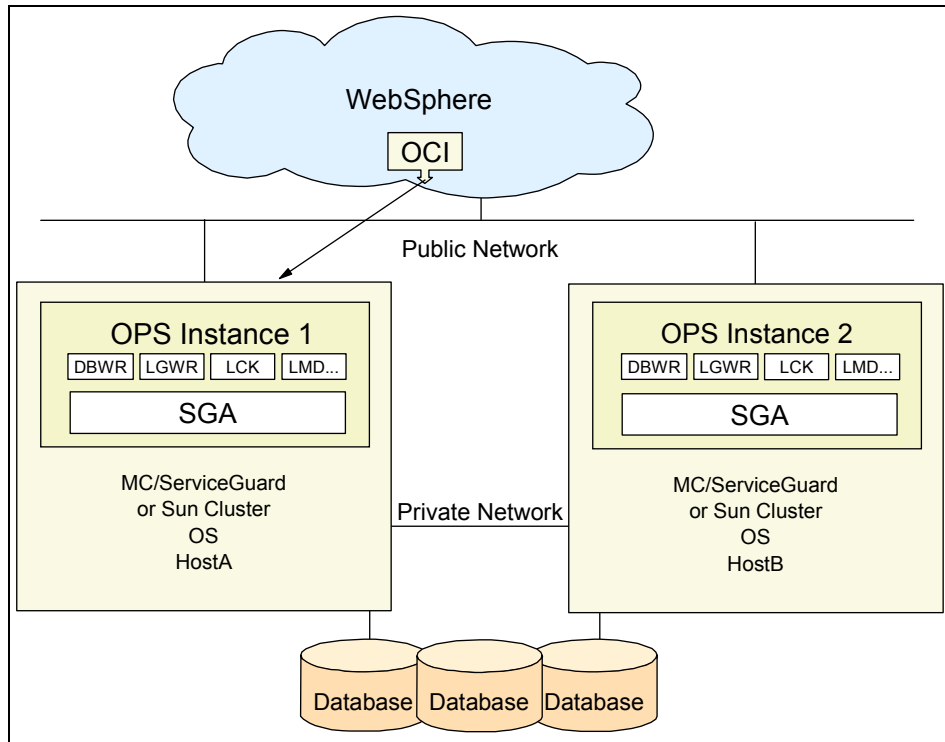


Figure 12-11 OPS/RAC configuration with load balancing

You can also configure OPS/RAC as a part of your Oracle distributed database system for your partitioned data or other applications, for the purpose of isolation or scalability. As shown in Figure 12-12 on page 482, four Oracle instances are created to serve three databases: Instance 1 and Instance 2 share the common data files and control files on multi-host shared disks.

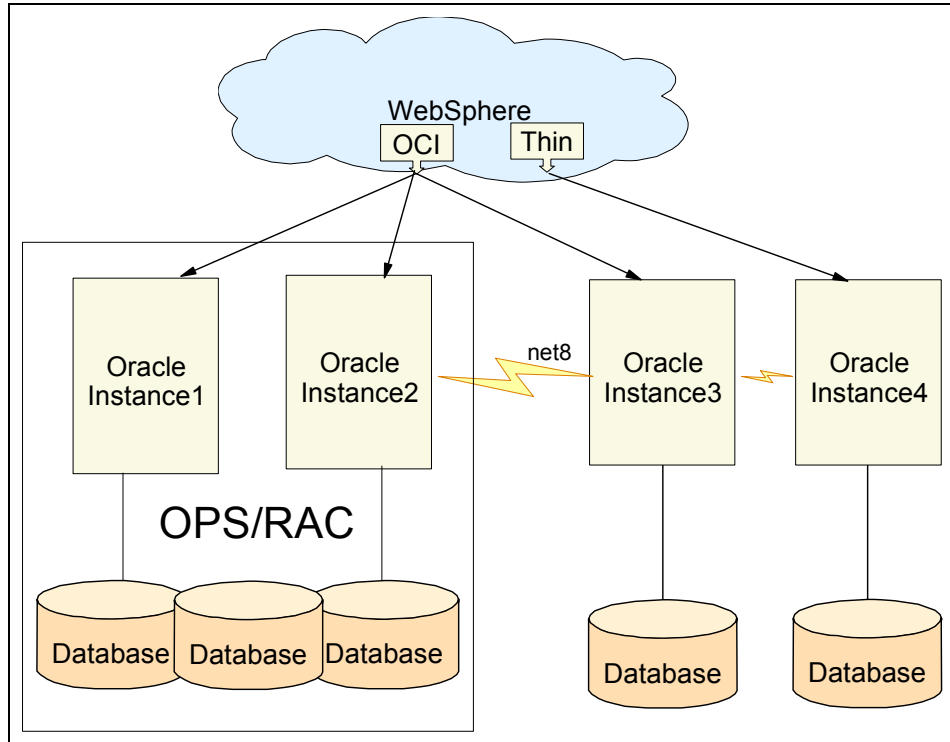


Figure 12-12 OPS/RAC and distributed database configuration

As shown in Figure 12-12, when an instance in OPS/RAC fails due to a failure of host hardware, software, network, or Oracle itself, OPS/RAC will automatically fail over to another pre-existing healthy Oracle instance. The healthy Oracle instance can access the log files of the failed Oracle instance and must complete cleanup work to maintain database integrity. Since OPS/RAC does not need to start another Oracle instance for failover, it is much faster than the IP takeover option.

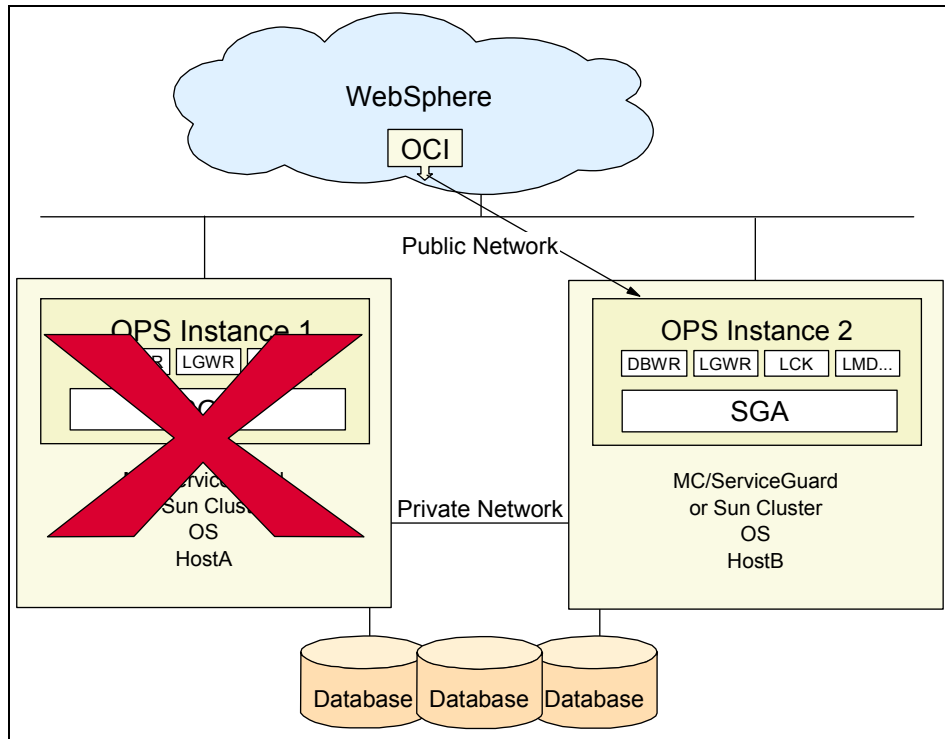


Figure 12-13 OPS/RAC after failover

We now describe the steps you have to follow to configure OPS/RAC for use with WebSphere.

### Installing and configuring the shared disk subsystem

OPS/RAC stores all common data such as logs, control files, and Oracle data in the shared disk that can be accessed by all OPS/RAC instances in different nodes. Each instance of an OPS/RAC database has its own log files that should be in the shared disk so that other instances can access them during recovery. Control files and data files are shared by instances in the cluster, so they are also put into the shared disk. You can install OPS/RAC binaries into the shared disk or into local disks of each node. We chose to install the binaries into the local disk of each node. The shared disk subsystem must be partitioned to be shared raw. The raw devices must be created by the root user.

## Creating raw devices

Create ASCII files with entries for each raw device file name for the Oracle Universal Installer.

Create an environment variable, `DBCA_RAW_CONFIG`, that points to the location of the ASCII file.

For Sun Cluster HA for Oracle Parallel Server disks, you can use either of the following configurations:

- ▶ VERITAS Cluster Volume Manager
- ▶ Sun Cluster Solstice Volume Manager

For MC/ServiceGuard Cluster HA for Oracle Parallel Server disks, you can use the following configuration:

- ▶ HPCluster Volume Manager

For other clustering software, see the corresponding section in this chapter.

## Installing and configuring the operating system-dependent layer

Install either Sun Cluster 2.2 or Sun Cluster 3.0 for Solaris, MC/ServiceGuard OPS Edition for HP UX, or MSCS. Configure the cluster software and diagnose the cluster configuration. See the corresponding section in this chapter for installation and configuration details, and for details on other clustering software.

## Installing and configuring OPS/RAC

Install Oracle 8i/9i Enterprise Edition, Net8 Server, Oracle Parallel Server/RAC Option, Oracle Intelligent Agent (if Oracle Enterprise Manager is used), Oracle Data Gatherer (if Oracle Diagnostics Pack is used for generating performance reports) and a Web browser if you have not installed it. The steps are:

1. Create the `dba` user group, then create the Oracle user and make it a member of the `dba` group.
2. Create a mount point directory on each node so that the name of the mount point on each node is identical, and so that the Oracle account has read, write, and execute privileges.
3. Grant permission to use the `rcp` command. Set up user equivalence on the node from which you will install OPS/RAC by adding entries for all nodes in the cluster to the `.rhosts` file of the Oracle account, or the `/etc/hosts.equiv` file.
4. Install Oracle 8i/9i Enterprise Edition with the OPS/RAC option. Please note that the Oracle Universal Installer will not make Oracle Parallel Server Option visible unless Cluster Manager cluster software has been properly configured.

If you have problems, please see the platform-specific cluster software configurations discussed previously in this chapter (Oracle integrated RAC with MSCS).

5. Select the required node in the cluster node selection window, for example opshp1 or opshp2. You will also be asked to enter a SID in the database identification window.
6. Follow the typical installation and create a database for WebSphere.
7. Start the listener of each node using **lsnrctl start**.
8. Start the database on each node using **connect internal/password; startup**.
9. Verify that all the defined OPS/RAC instances are running by using the following query:  

```
select * from v$active_instances
```
10. Fix any problems.

## Configuring OPS/RAC net service for failover

There are three files to be configured: listener.ora, tnsnames.ora, and sqlnet.ora. You can use **netasst** to configure them graphically, or you can just edit these files manually if you are confident with OPS/RAC. We directly edited these files, since it is faster to do so. You must manually edit the files to implement a new listener for failover and **netasst** does not allow using the same port number for another listener.

Implementing failover does not allow the use of static service configuration parameters in the listener.ora file. Therefore, in order to implement OPS/RAC failover, an additional listener must be specified on each node in the listener.ora file without service registration information. Edit the sqlnet.ora file to ensure that tnsnames.ora is used to resolve a net service name.

### ***Client load balance***

When you set `load_balance=on` in tnsnames.ora, OPS/RAC will balance the load over the list of listener addresses by picking one at random.

There are two different kinds of OPS/RAC failovers: connection-time failover (CTF) and transparent application failover (TAF). For TAF, there are several different configurations using TYPE and METHOD parameters.

### ***Connection-time failover (CTF)***

Connection-time failover refers to a client attempting to connect to a second listener when the attempt to connect to the first listener fails. To implement this, create a new net service name with the database name as its identifier, such as opswas, instead of opshp1 or opshp2. Put the address information of all nodes in the cluster in this newly created global net service name.

Use one of two options:

- ▶ **FAILOVER=ON**

Setting this means: try each address in order until one succeeds.

- ▶ **FAILOVER=ON LOAD\_BALANCE=ON**

Setting this means: try each address randomly until one succeeds.

WebSphere works with either setting.

Example 12-8 is a sample tnsnames.ora file using these OPS/RAC connection-time failover options.

*Example 12-8 Sample OPS/RAC CTF tnsnames.ora file*

---

```
WASOPSCTF.somecorp.com=
  (description=
    (load_balance=on)
    (failover=on)
    (address=
      (protocol=tcp)
      (host=idops1)
      (port=1521))
    (address=
      (protocol=tcp)
      (host=idops2)
      (port=1521))
    (connect_data=
      (service_name=WASOPSCTF.somecorp.com))))
```

---

### ***Transparent application failover (TAF)***

Transparent application failover enables an application to automatically reconnect to a database if the connection is broken. Active transactions roll back, and the new database connection is identical to the original one no matter how the connection is lost.

Implementing transparent application failover requires you to manually configure `tnsnames.ora` with the specification of a primary node, a backup node, failover type, and failover method. You can also combine OPS/RAC CTF with OPS/RAC TAF. As shown in Table 12-1, there are 12 configurations (2 CTF x 3 TYPE x 2 METHOD) for the OPS/RAC TAF implementations.

*Table 12-1 Combinations of OPS/RAC/TAF failover configuration implementations*

CTF	TYPE	METHOD
FAILOVER=ON	SESSION	BASIC
FAILOVER=ON LOAD_BALANCE=ON	SELECT	PRECONNECT
	NONE	

The parameter `TYPE` specifies the type of failover. This is a required parameter for the TAF failover. There are three options: `SESSION`, `SELECT`, and `NONE`. Selection of `SESSION` means that OPS/RAC fails over the session. A new session will be created automatically for use on the backup if a user's connection is lost. This type of failover does not attempt to recover selects after a failover. Choosing `SELECT` means that OPS/RAC allows users with open cursors to continue fetching on them after a failover. Although the performance for this type of TAF during a failover is good, it has an overhead during normal select operations. Selection of `NONE` means that no failover function is used. This is the default.

The parameter `METHOD` specifies how fast a failover occurs from the primary node to the backup node. There are two options: `BASIC` and `PRECONNECT`. Selection of `BASIC` means that OPS/RAC establishes connection at failover time. This option requires almost no work on the backup server until failover time. Selection of `PRECONNECT` means that OPS/RAC pre-establishes connections all the time. This is good for failover performance, but bad for normal operation, since it requires the backup instance to support all connections from every supported instance.

Example 12-9 shows a sample `tnsnames.ora` for OPS/RAC/TAF configuration with CTF.

*Example 12-9 Sample OPS/RAC TAF with CTF tnsnames.ora file*

---

```

WASOPSTAF.somecorp.com=
  (description=
    (load_balance=on)
    (failover=on)
    (address=
      (protocol=tcp)
      (host=opshp1)
    )
  )

```

```

        (port=1521)
    (address=
        (protocol=tcp)
        (host=opshp2)
        (port=1521)
    (connect_data=
        (service_name=WASOPSTAF.somecorp.com)
        (failover_mode=
            (type=session)
            (method=preconnect)
            (retries=0)
            (delay=10))))

```

---

Alternatively, Example 12-10 shows a sample tnsnames.ora for the OPS/RAC/TAF with primary and secondary servers configured.

*Example 12-10 Sample OPS/RAC TAF with primary/secondary servers tnsnames.ora file*

---

```

WASOPSTAF.somecorp.com=
    (description=
        (address=
            (protocol=tcp)
            (host=opshp1)
            (port=1521))
        (connect_data=
            (service_name=WASOPSTAF.somecorp.com)
            (failover_mode=
                (backup=opshp2)
                (type=session)
                (method=preconnect)
                (retries=0)
                (delay=10))))

```

---

WebSphere works with OPS/RAC/CTF failover configurations. WebSphere also works with some of OPS/RAC/TAF failover configurations, for example TYPE of SESSION, and METHOD of both BASIC and PRECONNECT.

## Configuring WebSphere servers

Install the Oracle client in the hosts where you plan to install WebSphere. Install WebSphere as described in the manual, using the OCI driver to connect the WebSphere Application Servers to the databases created. Since we use the global net service name, which is independent of individual nodes, the failover is transparent to WebSphere. For CTF, WebSphere receives a few `StaleConnectionExceptions` during an OPS/RAC failover, and WebSphere handles these exceptions automatically. WebSphere usually works automatically after an OPS/RAC/TAF failover.



In summary, although OPS/RAC uses non-IP failover, it still needs platform-specific cluster software to provide cluster management and inter-process communication. OPS/RAC failover takes a very short time. It is almost instantaneous failover, because non-IP takeover is used and no restarting of the database instance is needed. WebSphere establishes connections immediately after a failover, and clients observe very few failures.

## 12.9 WebSphere with DB2 Parallel Server

In this section, we look at integrating the WebSphere system with IBM DB2 Parallel Server. The topics discussed are:

- ▶ Introduction and considerations
- ▶ Setup and configuration

### 12.9.1 Introduction and considerations

Unlike Oracle OPS/RAC, which uses a shared data architecture, IBM DB2 UDB Enterprise Extended Edition uses a share-nothing architecture. Because data access congestion is avoided for a share-nothing architecture, IBM DB2 UDB EEE performs and scales well for very large databases.

IBM DB2 UDB EEE is the high-end edition for supporting clustered servers. It supports parallel query processing in both Symmetric Multiple Processing (SMP) and Massively Parallel Processing (MPP) environments needed for large data warehouse and high-volume online transaction processing (OLTP) systems. The major difference that distinguishes DB2 UDB EEE from other DB2 editions is the concept of a database partition. In DB2 UDB EEE, a database is not restricted to a single physical machine; a whole cluster of servers hosts a single database. The data stored in the database can be partitioned across the servers that participate in the cluster. Each partition stores a portion of the data in the database. SQL query processing is performed in parallel by all partitions. Query predicates that cannot be answered locally at a single partition are resolved among database partitions. Each row stored in a database on IBM DB2 UDB EEE is assigned to a specific partition. This assignment is based on the value of the partitioning key, which must be a subset of the primary key and every unique constraint for the table. A partitioning key is defined for a table using the CREATE TABLE statement. For details, please refer to the IBM DB2 UDB EEE documentation, since we focus on the availability issues here.

Since the data is partitioned across many machines, no single failure of machines or data partitions will cause the complete failure of the data service.

We can also enhance availability using IBM DB2 UDB partition failover controlled by clustering software such as HACMP, MC/ServiceGuard, Sun Cluster, VERITAS Cluster, or MSCS, as shown in Figure 12-14. We have three partitions on three machines, and the fourth machine is used as a hot standby. If any of the three partitions fails, it will fail over to the hot standby machine.

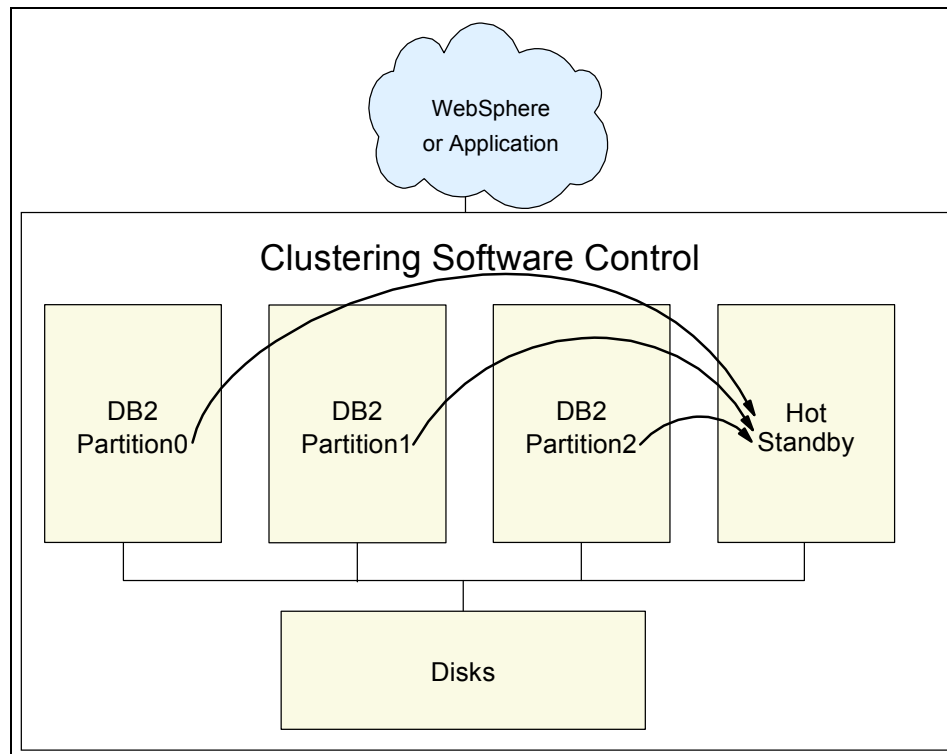


Figure 12-14 DB2 Parallel Server partition failover under control of cluster software

## 12.9.2 Setup and configuration

The configuration process is similar to that described in the previous sections in this chapter for clustering hardware, networks, and software. We highlight the differences here.

WebSphere uses two databases: the persistent sessions database for HTTP sessions, and the application database for Entity EJBs and others.

The HTTP persistent sessions database usually does not contain enough data to benefit from database partitions. The small overhead of coordinating partitions in a DB2 UDB EEE instance may incur a minor performance penalty. However, the application database may be big enough for many customers to benefit from database partitions across several nodes in a cluster.

Since the WebSphere application database is a multiple-partition database and physically resides on multiple nodes, the steps for defining the corresponding data source are different. The custom properties are needed to define the coordinator node (cnode). This is a DB2-specific connection attribute for DB2 UDB EEE. We chose the first node as a coordinator node (cnode=0) to catalog. You can choose any node as a coordinator node, for example cnode=0, cnode=1 or cnode=2.

You must modify the generated DDL file for container-managed persistence (CMP) EJBs developed using the "top-down" approach, because the generated DDL does not define the partition key required for DB2 UDB EEE.

You need to modify the WebSphere application deployment generated table DDL to include the partitioning key. The modified sample DDL file is shown in Example 12-11.

*Example 12-11 Sample DDL file modified for DB2 UDB EEE partitioning*

---

```
CREATE TABLE MyCMPBean (  
    group INTEGER,  
    country VARCHAR(3) NOT NULL,  
    name VARCHAR(20) NOT NULL,  
    phone VARCHAR(12) NOT NULL,  
    age INTEGER,  
    city VARCHAR(15))  
    partitioning key (country) IN TABLESPACE ptb);  
ALTER TABLE MyCMPBean ADD Constraint MyCMPBeanPK PRIMARY KEY (country, name,  
phone);
```

---

Now we have defined the partitioning key as country, a part of the primary key. Remember that the partitioning key must be a subset of the primary key or you will get an error. It is very important to choose the partitioning key, since it can impact performance. Partitioning the database by country, incidentally, is a common practice.

As listed in Table 12-2 on page 492, each partition runs in its first preference host if there are no failures, and the configuration file looks like this:

```
0 host1 host1 0 10.1.1.11  
1 host2 host2 0 10.1.1.12  
2 host3 host3 0 10.1.1.13
```

Table 12-2 Failover preference of DB2 partitions

DB2 Partition	First Preference	Second Preference
DB2 Partition 1	Host1	Host4
DB2 Partition 2	Host2	Host4
DB2 Partition 3	Host3	Host4

When host2 fails, the partition 2 is moved to host4 with the same IP address and the same disks as:

```
0 host1 host1 0 10.1.1.11
1 host4 host4 0 10.1.1.12
2 host3 host3 0 10.1.1.13
```

Similarly, we can configure multiple partitions in a host, and mutual takeover. If more than one partition uses a host, then different ports are used to avoid any conflicts. The simplest configuration is to configure two or more partitions in a two-node system with mutual failover.

Please note that the failover discussed here for DB2 UDB EEE is an IP-based failover that takes over the resource group and IP address. DB2 UDB EEE itself doesn't provide HA for data access since the share-nothing architecture is used. With the assistance of such clustering software as HACMP, ServiceGuard, VERITAS Cluster, or Microsoft Cluster, DB2 UDB EEE can provide both scalability and high availability.

## 12.10 WebSphere and non-IP-based database failover

For non-IP database failover such as Oracle Parallel Server (OPS) and Real Application Cluster (RAC), the failover process is much faster, usually finishing in seconds. Therefore, very few failures can be observed. This type of HA data management is especially suitable for the WebSphere persistent session data. Some transparent failover techniques that database vendors use further reduce the failures clients may observe.

The failed database instance may cause data inconsistency. This data inconsistency is repaired by starting a new instance and performing crash checks in the IP-takeover situations discussed above. However, since the backup database instance is pre-existent, a new instance does not need to be started in the non-IP-takeover situation. The pre-existent healthy database instance must repair any data inconsistency problems left by the failed instance.

Compared with IP-based failover (within one to five minutes), non-IP-takeover failover is very fast (within seconds). Therefore, clients see very few failures.

## 12.11 One instance for all WebSphere databases?

The WebSphere system uses several databases for different purposes: persistent session data, application data including EJBs, MQ data, and LDAP data.

We can place all these databases in a single instance, in multiple separate instances, or in grouped multiple instances for failover.

Fewer database instances make a system simpler and easier to administer. However, hosting multiple databases in an instance implies that these databases must be hosted in the same machine. This may conflict with the goal of scalability and availability. Each instance has an independent TCP/IP listener process, so separate listeners may also boost performance. Furthermore, separate instances allow fine-tuning of individual instance configuration parameters for performance. Separate instances may also increase the flexibility of instance monitoring and repair: one instance can go down without breaking everything.

Having all databases in a single database instance, as shown in Figure 12-15, is not good for either scalability or availability.

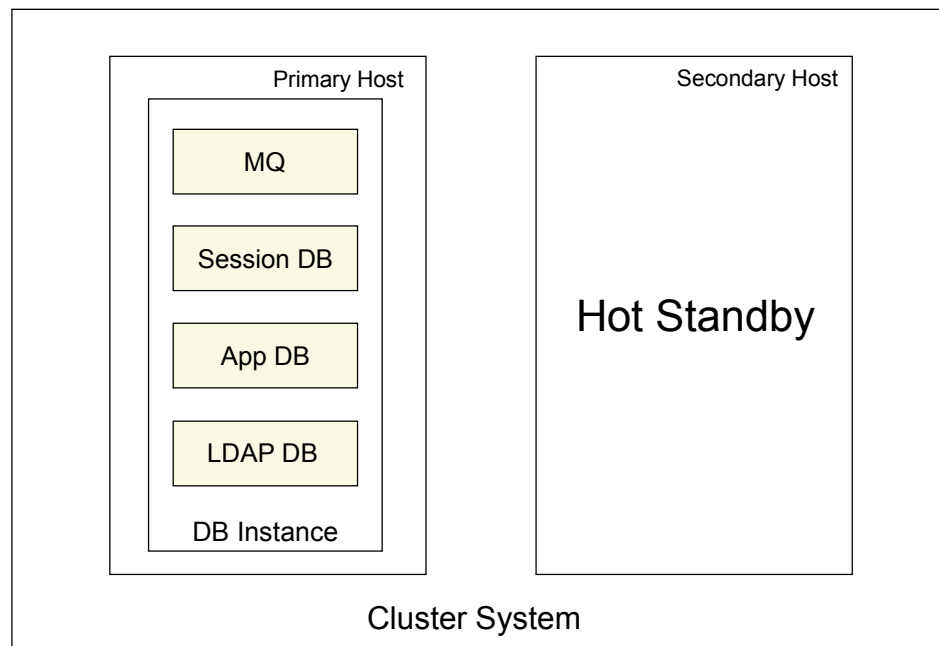
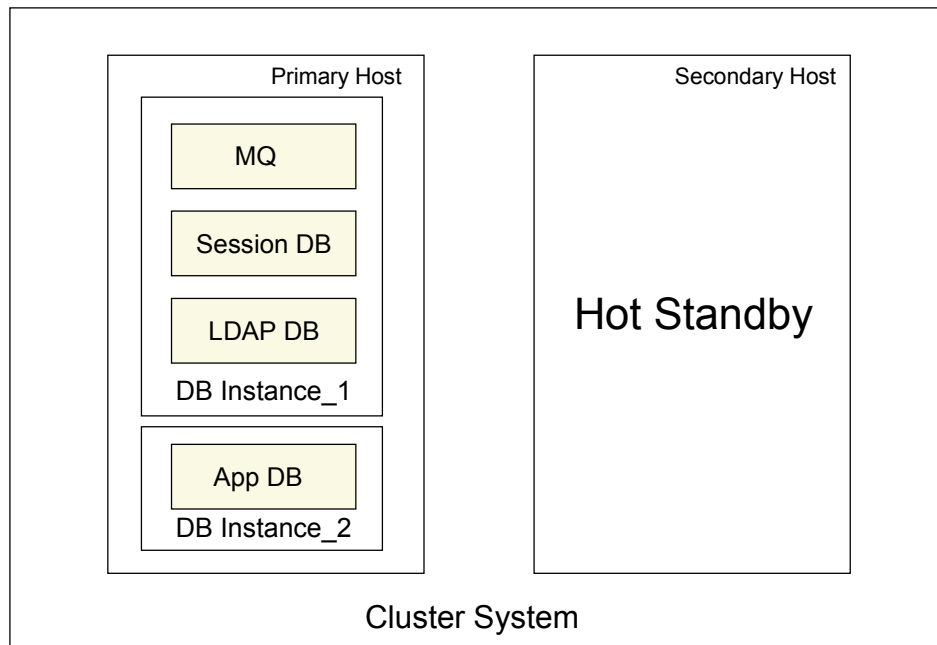


Figure 12-15 Single DB instance with hot standby

A slightly better configuration separates the application database (Entity EJBs and other application data) from the MQ repository, HTTP persistent session database, and LDAP database, as shown in Figure 12-16.



*Figure 12-16 Separate application database with hot standby*

As shown in Figure 12-17 on page 495 and Figure 12-18 on page 496, separating the LDAP database and the session database from the MQ repository provides further freedom to use HA solutions.

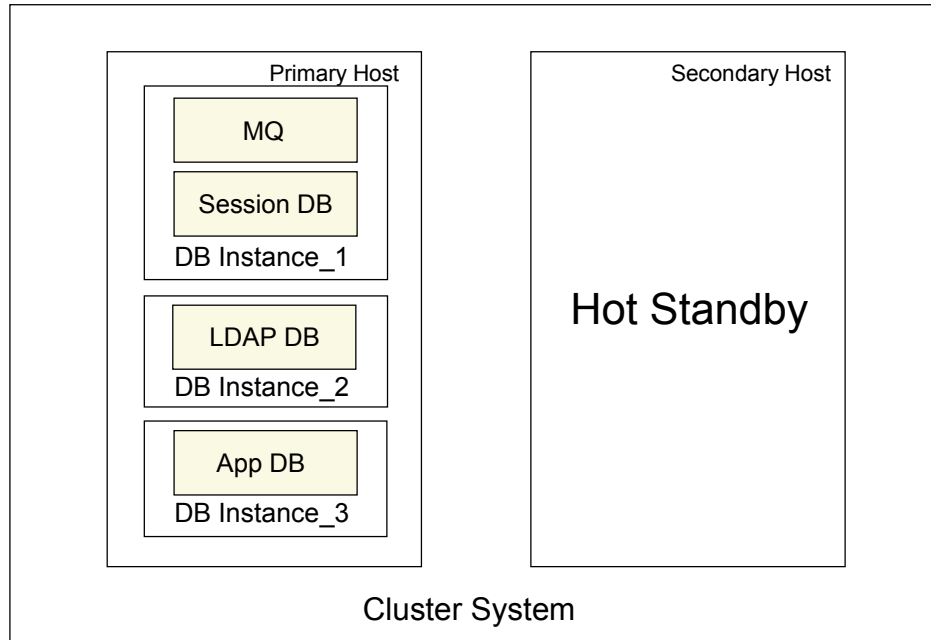
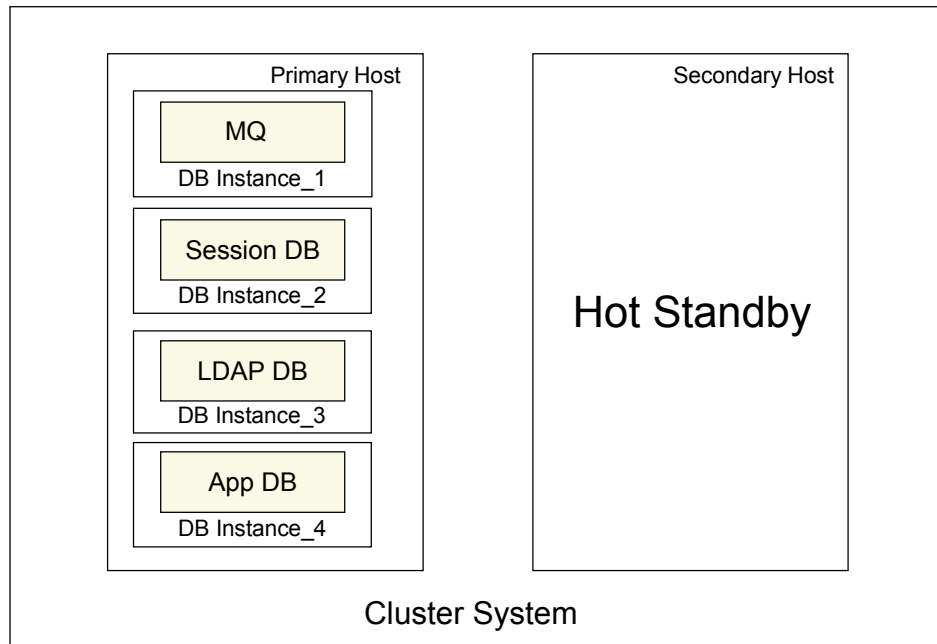


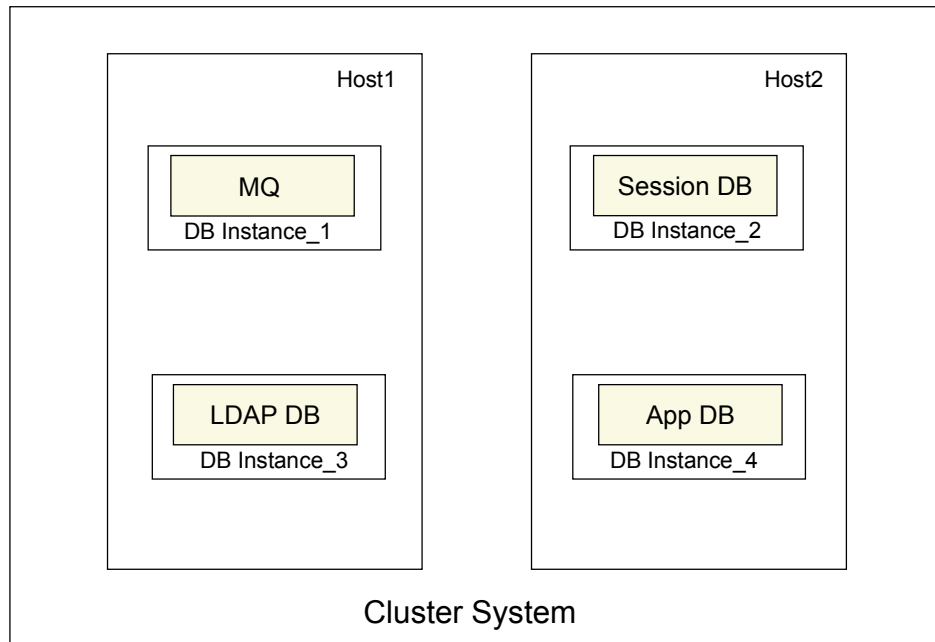
Figure 12-17 Separate LDAP and application databases with hot standby



*Figure 12-18 Separate all databases with hot standby*

We can use the second machine under a cluster to host the session database and the application database to balance the load, as shown in Figure 12-19 on page 497.





*Figure 12-19 Mutual takeover of different database instances*

We also can build separate clusters specially for the application database and other databases, as shown in Figure 12-20 on page 498.

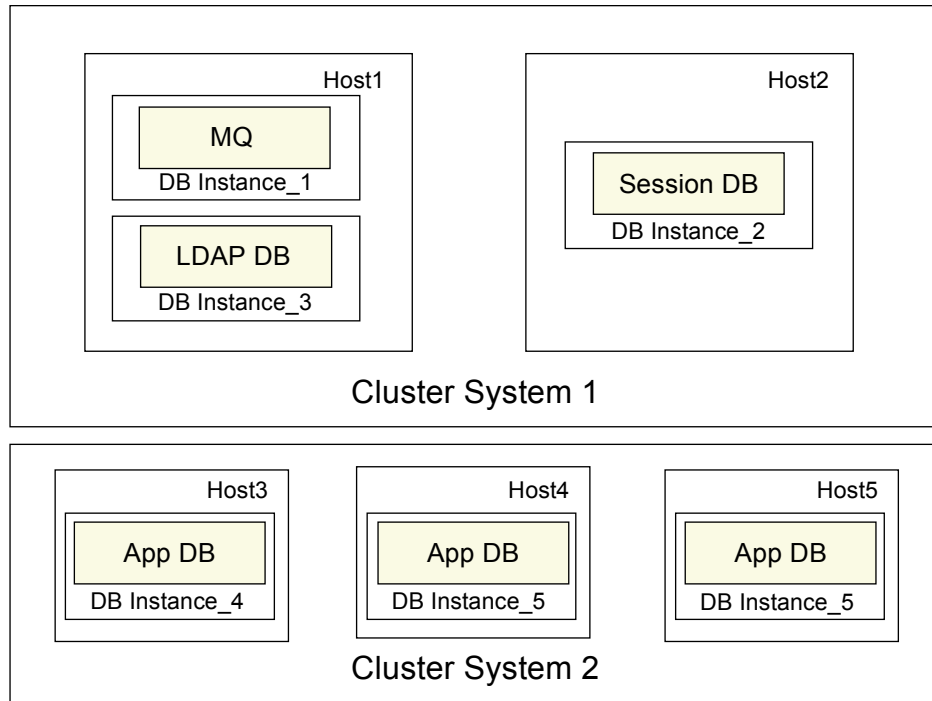


Figure 12-20 Different clustering system for application databases

## 12.12 WebSphere and Web server high availability

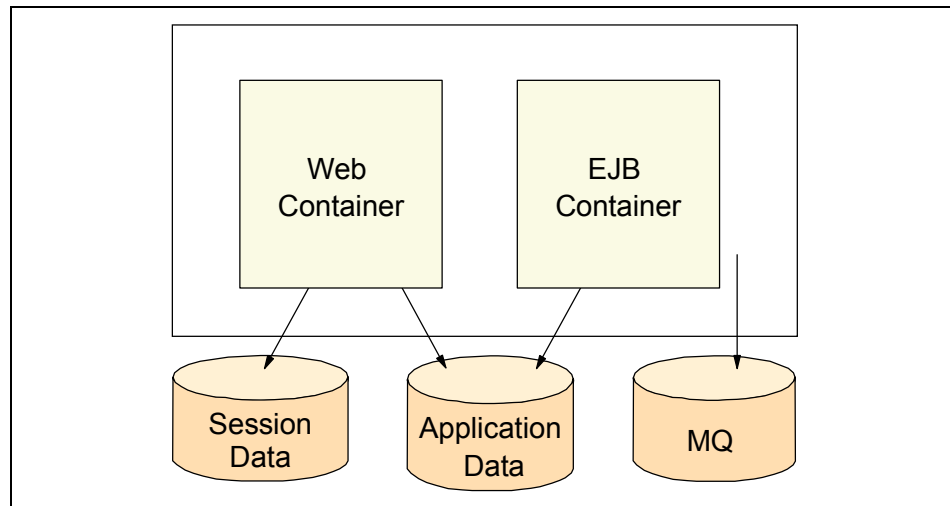
In this section, we discuss WebSphere system availability, including both process and data availability. WebSphere system availability includes WebSphere Application Server and administrative server process availability, as well as data availability. WebSphere availability depends on end-to-end system availability. Any break in the chain can break the whole system and make the WebSphere service unavailable.

WebSphere server process availability is achieved through the WebSphere workload management (WLM) mechanism, while data is made highly available through clustering techniques. The data may include Entity EJB data, servlet session data, and administrative data, including naming and security.

Depending on whether the Web container and EJB container are on the same host and whether vertical and/or horizontal scaling (WLM) is/are used, we can have several configurations with different levels of availability:

- ▶ Single host with single Web container and EJB container, shown in Figure 12-21.
- ▶ Single host with multiple Web containers and multiple EJB containers, shown in Figure 12-22 on page 500.
- ▶ Multiple hosts with Web container(s) and EJB container(s) in the same hosts, shown in Figure 12-23 on page 500.
- ▶ Multiple hosts with Web container(s) and EJB container(s) in different hosts, shown in Figure 12-24 on page 501.

Our discussions on the service availability of these configurations assumes that we have HA databases, with session data persisted to an HA database.



*Figure 12-21 Web and EJB container in a single host*

When the host dies, the WebSphere service will be unavailable for the configuration shown in Figure 12-21. When either the Web container process or EJB container crashes, the WebSphere service will be unavailable. When the network breaks, the WebSphere service will be unavailable.

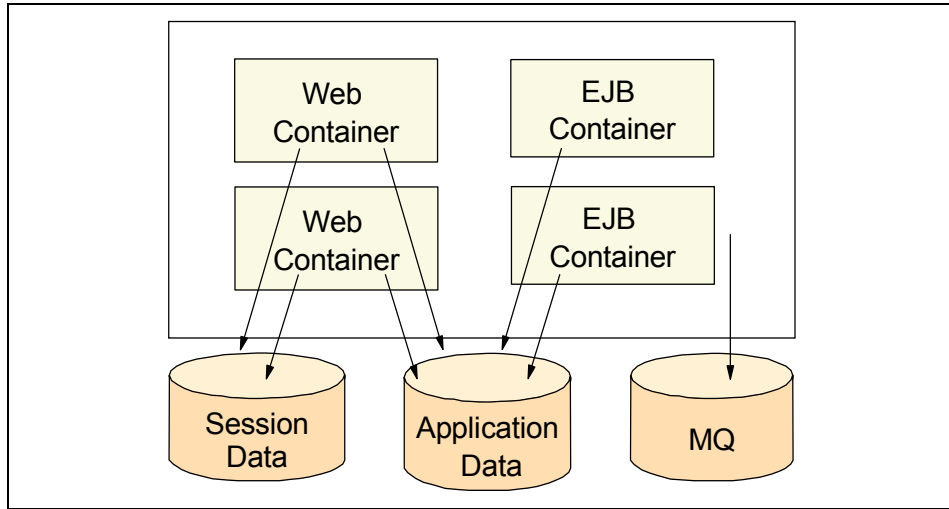


Figure 12-22 Multiple Web and EJB containers in a single host

When the host dies in the configuration shown in Figure 12-22, the WebSphere service will be unavailable. When the network breaks, the WebSphere service will be unavailable. However, when the Web container process and/or the EJB container crashes, the WebSphere service is still available, as long as not all Web containers and EJB containers crash.

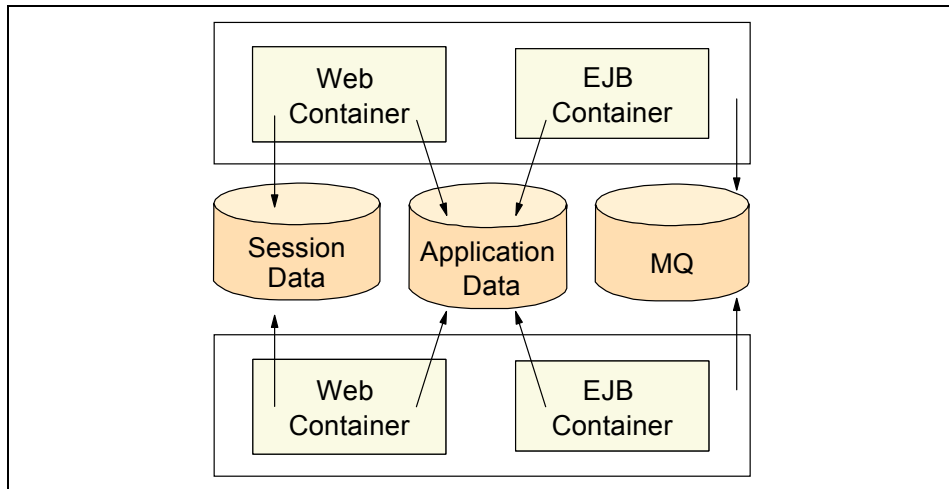
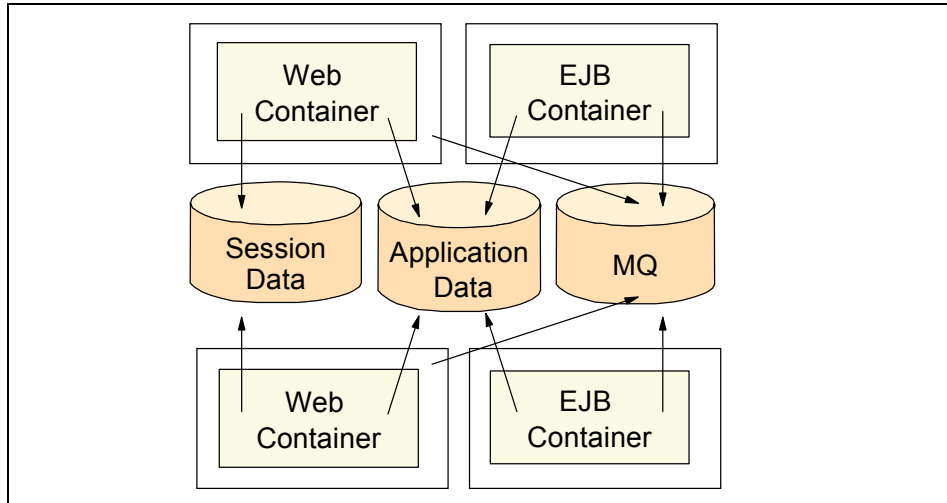


Figure 12-23 Multiple Web and EJB containers in multiple hosts

The configuration shown in Figure 12-23 tolerates both process faults and host faults.



*Figure 12-24 Separate Web and EJB containers in multiple hosts*

The configuration shown in Figure 12-24 enhances system availability by separating the Web container(s) and EJB container(s). We can also further enhance WebSphere availability by using multiple WebSphere administrative domains so that hardware and software upgrades will not make the system unavailable.





## High availability of LDAP, NFS, and firewall

As shown in Figure 13-1 on page 504, a complete WebSphere system includes several components. WebSphere system availability is determined by the weakest part in the chain, as discussed in Chapter 8, “High availability concepts” on page 315. Therefore, in this chapter, we discuss how to make the LDAP server, the networked file system server, and the firewall server highly available in a WebSphere system.

The content of this chapter has not changed since the previous version of this redbook.

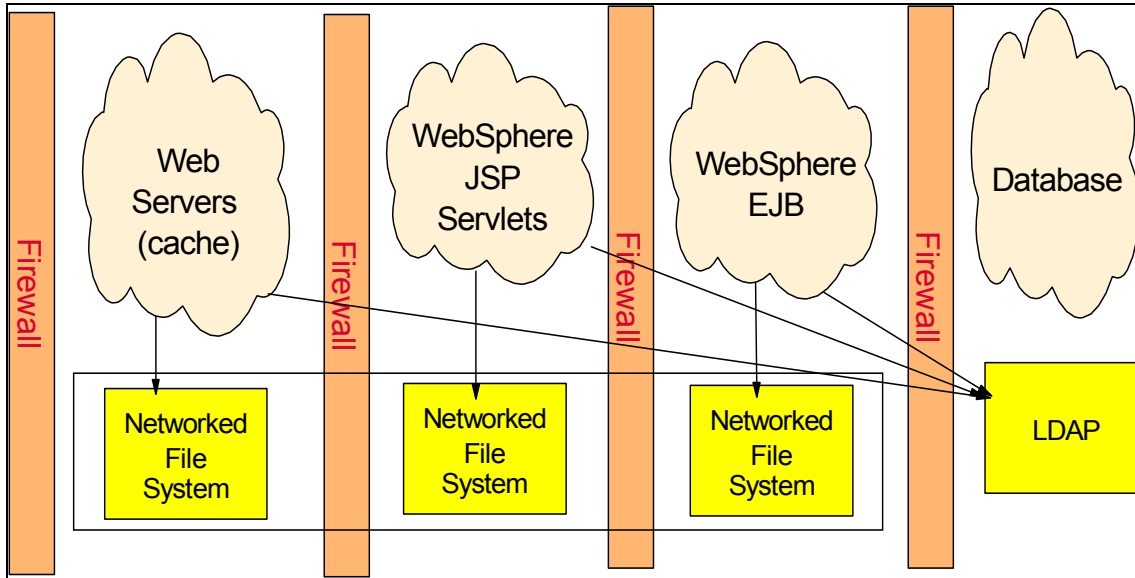


Figure 13-1 WebSphere system high availability - HA roles of LDAP, firewall, networked file system, and Web server load balancer in a WebSphere system

## 13.1 Load Balancer high availability

The Load Balancer (LB) from the IBM WebSphere Application Server Network Deployment V5.1 Edge Components is used to spray network traffic for both load balancing and high availability. In order to avoid a single point of failure, the Load Balancer should be highly available. This can be achieved by setting up a second LB server with exactly the same configuration. You have to define one or more heartbeat connections between these servers. The heartbeat mechanism is built into the WebSphere Load Balancer. You can configure a two-node LB as either active/active mutual takeover or active/passive failover. We used the active/passive failover configuration shown in Figure 13-2 on page 505 in our end-to-end WebSphere production system. The Load Balancer from the WebSphere Edge Components is used to provide high availability for Web servers and firewalls in our configuration. For details on LB configuration, see Chapter 4, “Web server load balancing” on page 93, the redbook *WebSphere Edge Server: Working with Web Traffic Express and Network Dispatcher*, SG24-6172, and the redbook *Patterns for the Edge of Network*, SG24-6822.



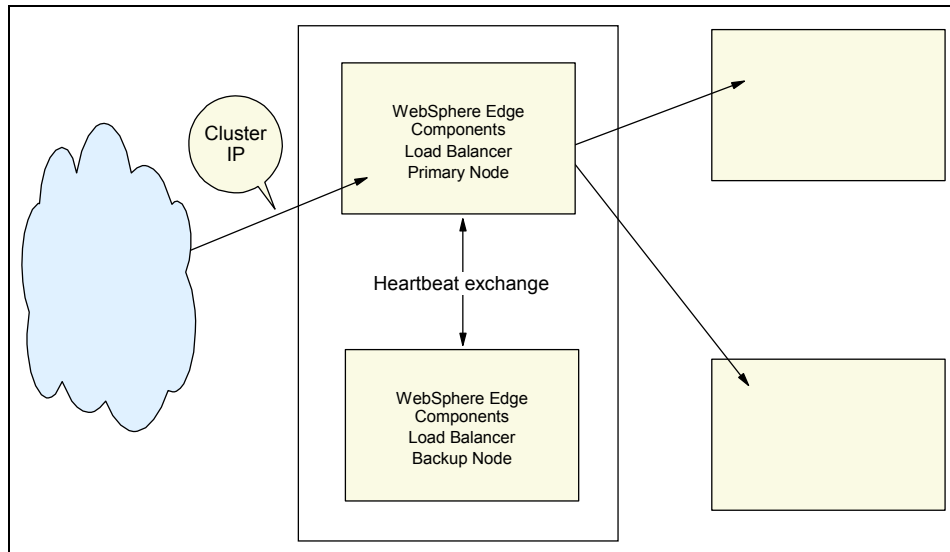


Figure 13-2 Load Balancer configuration

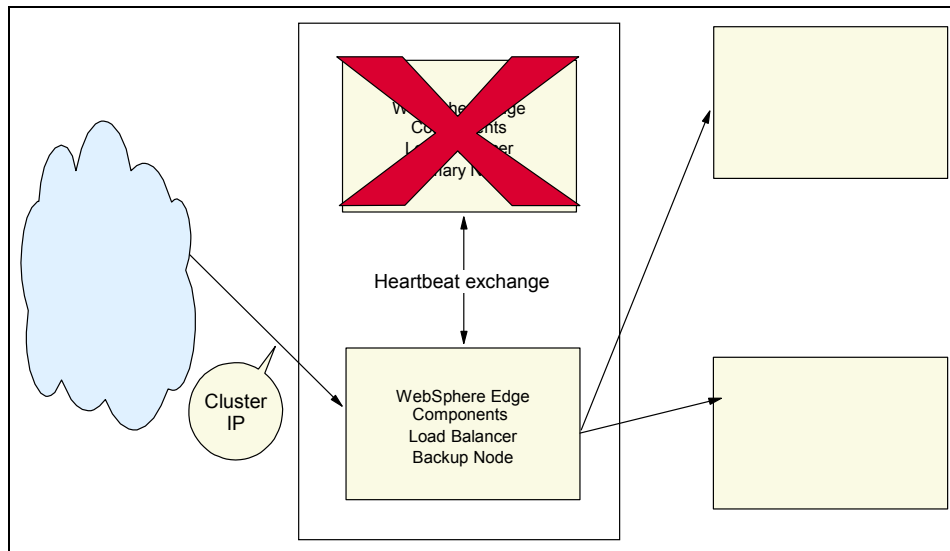


Figure 13-3 Load Balancer configuration after failover

There is constant synchronization between the two servers, so the backup server knows which connections are still active. In addition, you can specify several reach targets, which are network devices that are constantly pinged by the LB in order to detect network failures. A takeover will be issued for one of the following reasons:

- ▶ The heartbeat connection has been interrupted. This indicates a hardware failure on the primary LB server, and the backup server issues a takeover. If this was simply a network failure, the backup switches back into standby mode, if the heartbeat from the primary reaches the backup.
- ▶ Both servers constantly ping all reach targets. If the backup server can reach more targets than the primary LB server, there must be a network failure on the primary server, and a takeover will be issued. Since there is still a heartbeat between the two machines, the primary server is informed about this event and switches to standby mode, so there is no IP address overlap.
- ▶ If the backup server is active and the primary server is on standby, and the backup server has a hardware failure (no heartbeat), the primary server immediately switches into active mode again. As you can see, the heartbeat is a very important indicator; losing this heartbeat indicates a complete hardware failure. Therefore, it is better to have multiple heartbeat connections, for example on every network interface, to ensure that a network failure on one interface does not result in the loss of the heartbeat connection. In addition, you should have a reach target in every network; these are used to determine if a specific network connection is broken. There are some external scripts executed by the LB in the case of a status switch. All these scripts must be placed in the `/usr/lpp/nd/dispatcher/bin` directory:
  - `goActive`: this script indicates that the LB will switch into active mode and start dispatching packets. The script must ensure that the cluster IP address is configured correctly on the network card. This script is executed by the primary LB server, if there is no active backup, or by the backup LB server, if there is a takeover.
  - `goStandby`: this script indicates that this server will stop routing packets. This happens if the active LB server has a problem with one of its network cards, and the backup server takes over. At this time, the primary server should make sure that the cluster IP address is no longer advertised on the network.
  - `goInOp`: this script is executed when the Executor is stopped, and it should clean up the system.

Of course, these scripts are used to reconfigure the network interfaces if the high availability feature of LB is used directly on the firewall machines.

There is third-part hardware/software available with some functions of Load Balancer, for example Cisco director or DNS server. Some of them have no full failure-detection function.

## 13.2 LDAP high availability

A directory is often described as a database, but it is a specialized database that has characteristics that set it apart from general-purpose relational databases. One special characteristic of directories is that they are accessed (read or searched) much more often than they are updated (written). Lightweight Directory Access Protocol (LDAP) is a fast-growing technology for accessing common directory information. LDAP has been embraced and implemented in most network-oriented middleware. Building LDAP-enabled networks and applications is a common practice in enterprise applications. WebSphere is LDAP-enabled. When the LDAP server fails, WebSphere cannot access directory data, such as security data, and hence fails to service client requests. Therefore, building HA LDAP is a part of the highly available WebSphere system, as shown in Figure 13-4 on page 508.

In a way similar to building an HA database for WebSphere, you can build an HA LDAP service with clustering software such as HACMP, MC/ServiceGuard, or Microsoft Cluster Service, as shown in Figure 13-4 on page 508. Two nodes are interconnected with public networks and private networks. Private networks are dedicated to the heartbeat message. A shared disk that is connected to both nodes is used to store common directory data. Clustering software and LDAP software are installed in each node. A resource group is created, and can be failed over from one node to the other under the control of the clustering software. Instead of individual physical host IP addresses, the cluster IP address is used to access the LDAP service.

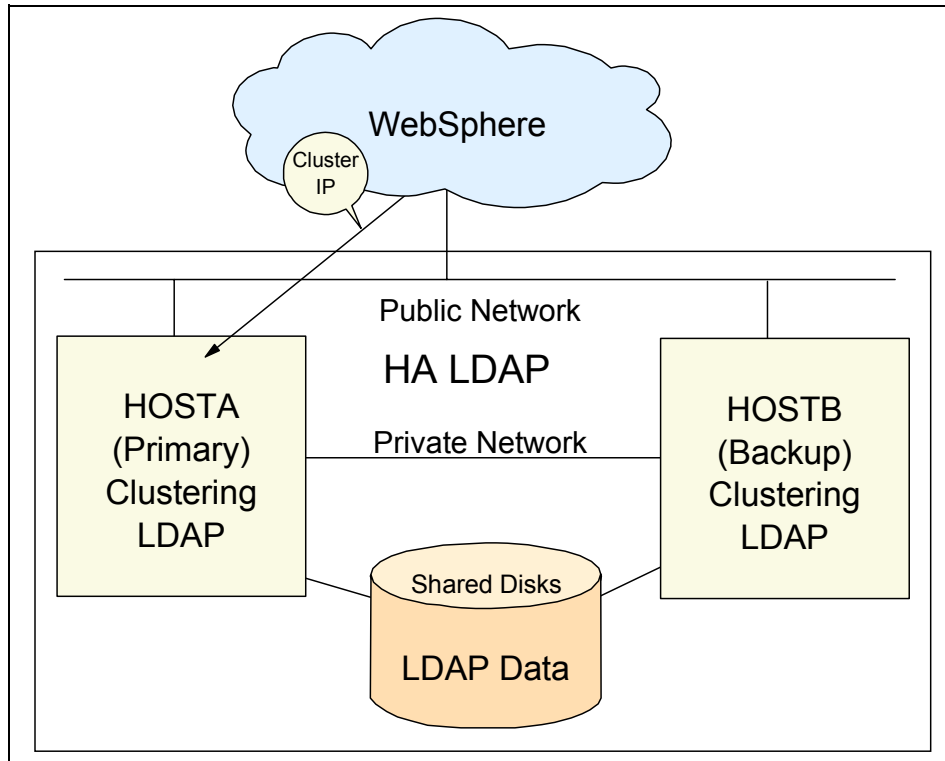


Figure 13-4 Clustered LDAP with shared disks

The cluster IP address is moved to the healthy backup node under the control of the clustering software when the primary node fails and the cluster detects the failure through the heartbeat mechanism. The LDAP client (WebSphere) still uses the same IP address (the cluster IP address) to access the LDAP service, as shown in Figure 13-5 on page 509. You can configure the service to fall back automatically to the primary node once the primary node is up again, or you can do it manually.

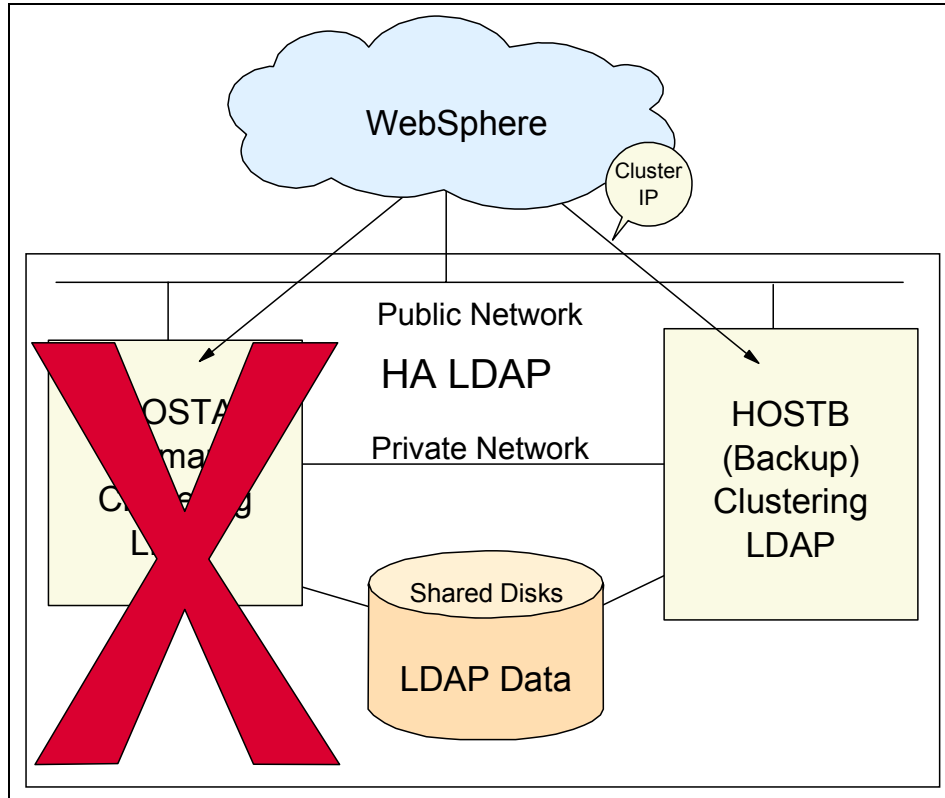


Figure 13-5 Clustered LDAP with shared disks after failover

The multihost shared disk is used in the above configuration for storing LDAP data. In addition, LDAP provides a master and replica architecture that makes it possible for you to configure HA LDAP without shared disks. Install clustering software on both nodes, and configure LDAP to use local data. The primary node is configured as the LDAP master, and the backup node is configured as the LDAP replica, as shown in Figure 13-6 on page 510. Any LDAP change requests that go to the replica server will be referred to the master server, since the replica server cannot change data. The master server sends all changes to the replica server to synchronize its data.

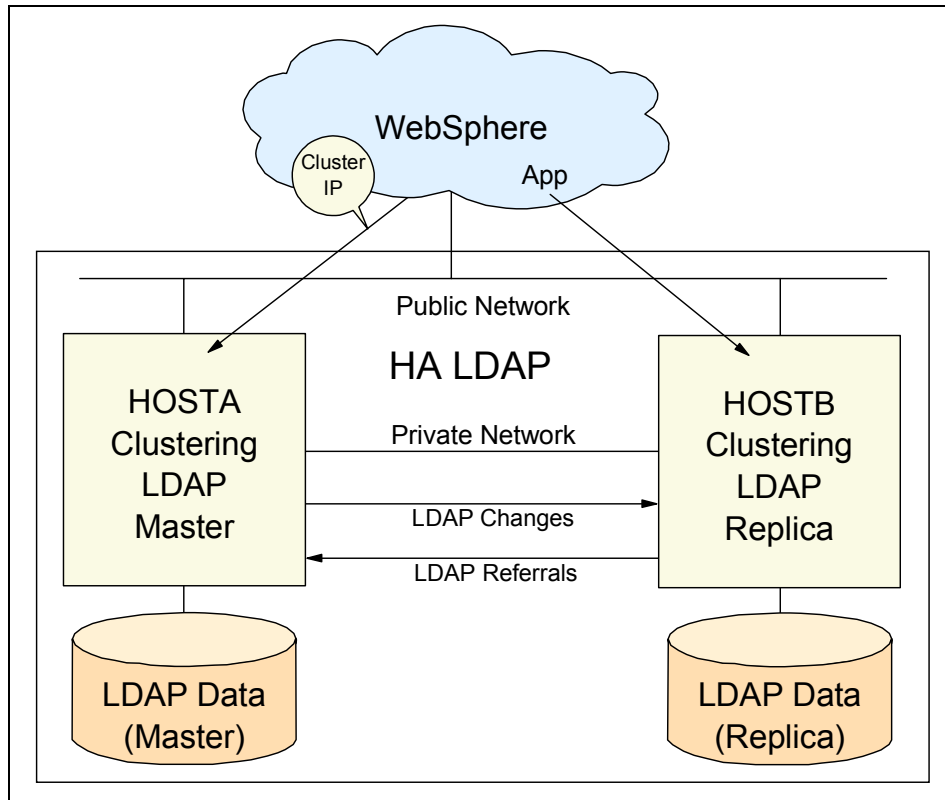


Figure 13-6 Clustered master-replica LDAP without shared disks

When the primary server (master) is down due to some network, hardware, or software reason, the LDAP service will be moved to the backup server under the control of clustering software; the replica server is temporarily promoted to the master server and continues LDAP service, as shown in Figure 13-7 on page 511.

When the primary node is up again, you can move the LDAP service back to the primary node. You should not configure automatic fallback, because by doing so, you will lose all updates. You need to manually export the latest data from the backup server and import it to the primary server before you start up the primary LDAP server. It takes time to synchronize the data in the master server in this share-nothing configuration. In the shared disks LDAP configuration, since you use the same data in the shared disks, you do not need to synchronize the data between two servers. However, it is easier to configure the cluster without shared disks.

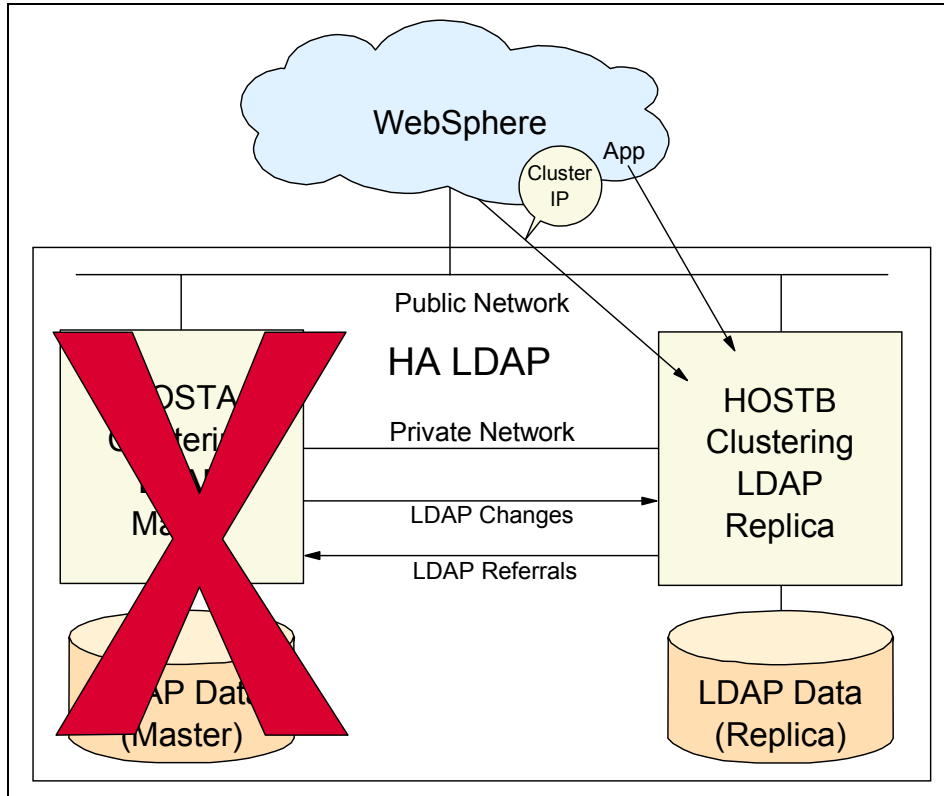
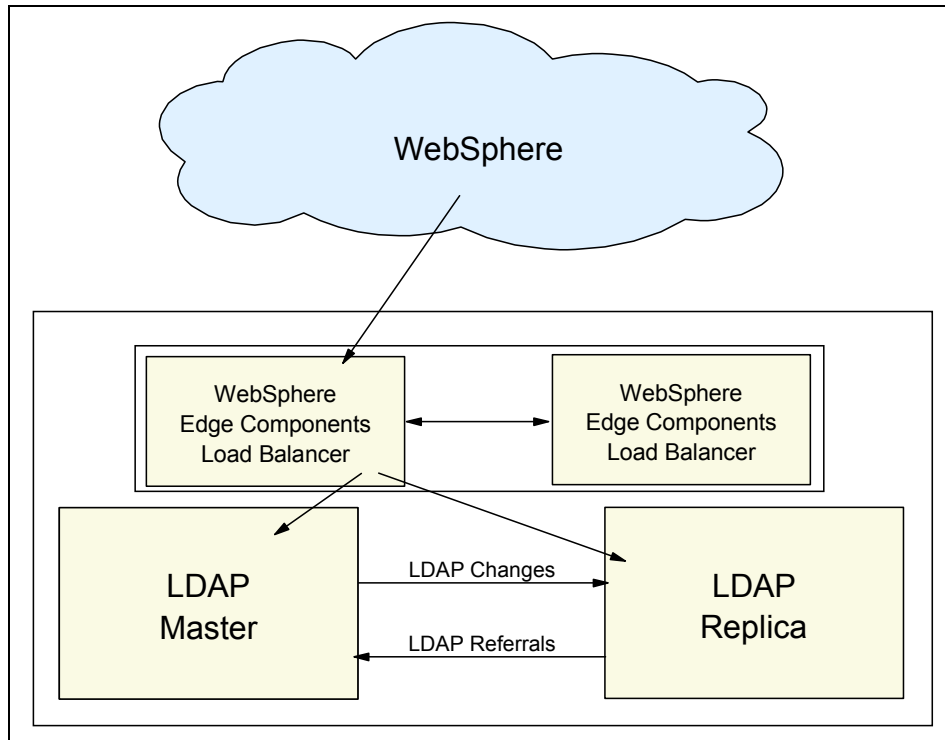


Figure 13-7 Clustered master-replica LDAP without shared disks after failover

In addition to HA LDAP with clustering software, you can build a low-cost, easy-to-configure HA LDAP with a network sprayer such as the WebSphere Edge Components' Load Balancer, or a DNS server that has a load balancing function (DNS round robin), as shown in Figure 13-8 on page 512.



*Figure 13-8 LDAP and Load Balancer*

The Load Balancer distributes client requests to both servers. When a server fails, the request will be directed to the other server, as shown in Figure 13-9 on page 513.



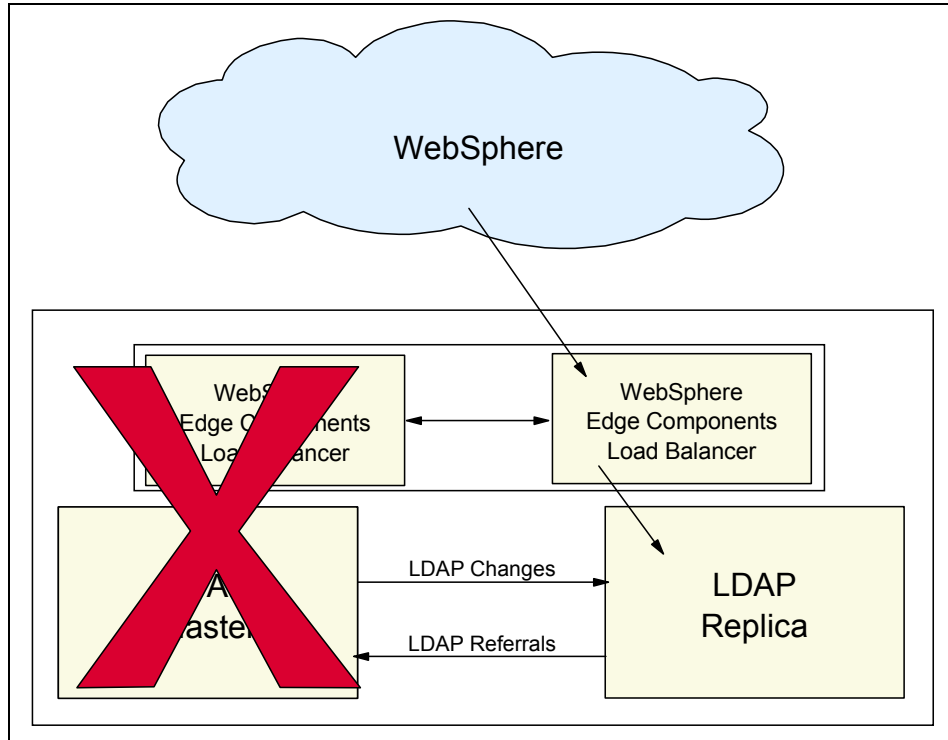


Figure 13-9 LDAP and Load Balancer after failover

For high-end enterprise applications, combined clustering software and a network sprayer can improve LDAP availability and scalability by reducing downtime and providing more servers, as shown in Figure 13-10 on page 514. During the clustering transition downtime, you can still access LDAP servers (read only) with this configuration. You can also partition your directory structure to enhance scalability, and use approaches discussed here to enhance availability.

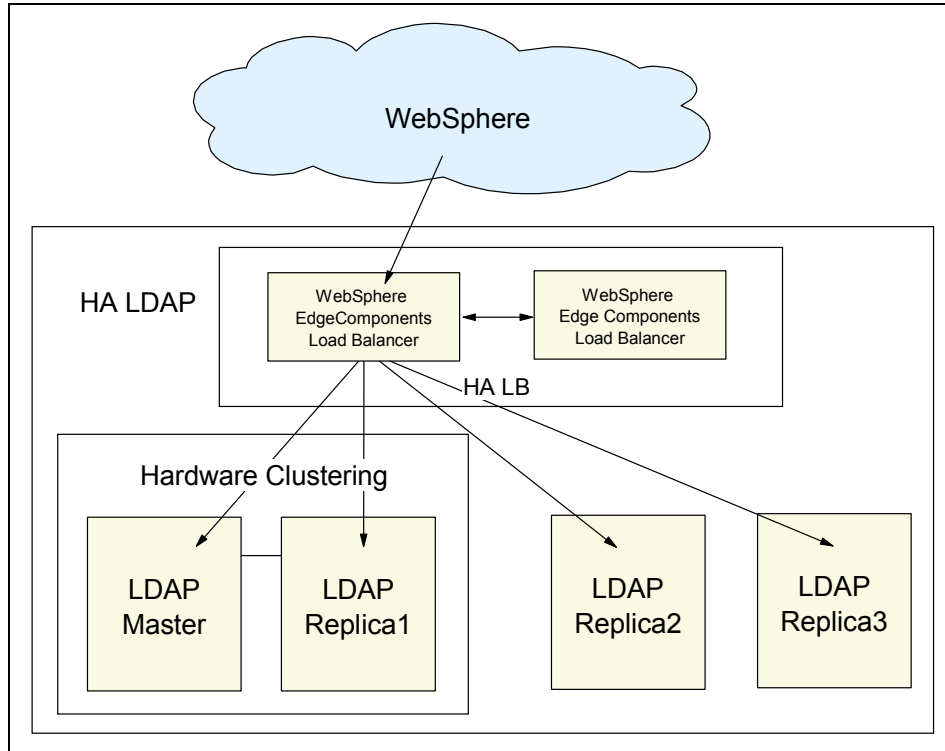


Figure 13-10 Combined LB and clustered LDAP

## 13.3 Firewall high availability

Usually, a WebSphere production system includes at least one firewall. Two firewalls are commonly used to create a demilitarized zone (DMZ) to enhance WebSphere system security. If the firewall fails, customers are not able to access any services and the site can be exposed to security risks (hacker's attacks). Therefore, the firewall availability is an important part of the WebSphere system's availability.

We can configure a highly available firewall environment, such as IBM SecureWay® eNetwork firewall or CheckPoint VPN-1/FireWall-1, by using two separate firewalls on two hosts. The CheckPoint VPN-1/FireWall-1 product provides several HA features, such as the state synchronization of the firewall modules that allow active connections to continue after failover. However, there is no built-in mechanism in VPN-1/FireWall-1 to synchronize the security policy (filter rules and users) across two VPN-1/FireWall-1 management stations. The VPN-1/FireWall-1 management workstation is a single point of failure.

In this section, we discuss two advanced solutions:

- ▶ Building an HA firewall with clustering software such as HACMP, MC/ServiceGuard, Sun Cluster, or MSCS.
- ▶ Building an HA firewall with a network sprayer such as WebSphere Edge Components' Load Balancer.

### 13.3.1 Using clustering software

As shown in Figure 13-11, clustering software such as HACMP, MC/ServiceGuard, or MSCS is used to provide highly available service IP addresses, resource groups, and fault-monitoring mechanisms. For the clustering configuration, please refer to previous sections in this chapter.

Each node has a complete installation of IBM SecureWay eNetwork Firewall, CheckPoint FireWall-1, or another firewall. Configure both nodes in such a way that the equal and interchangeable configurations on both nodes are assured.

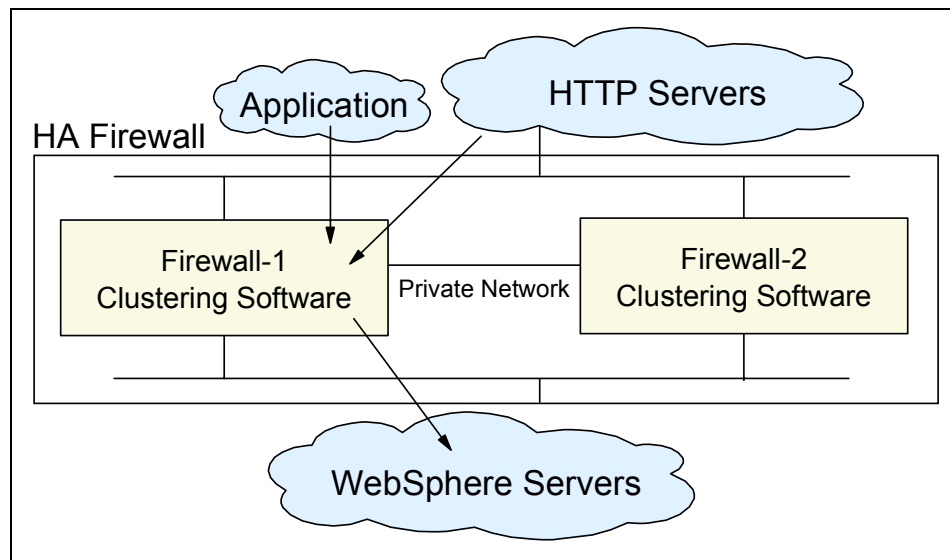


Figure 13-11 Clustered firewall for high availability

When the firewall process, network, or machine itself goes down, the clustering software will detect it and relocate the resource group, including the service IP address, to the backup node, then start the firewall service, as shown in Figure 13-12 on page 516. The highly available firewall environment is reestablished after a failover. As soon as the primary firewall is up, the firewall service can automatically fall back to the primary node, or you can do this manually (this is determined by the clustering configuration settings).

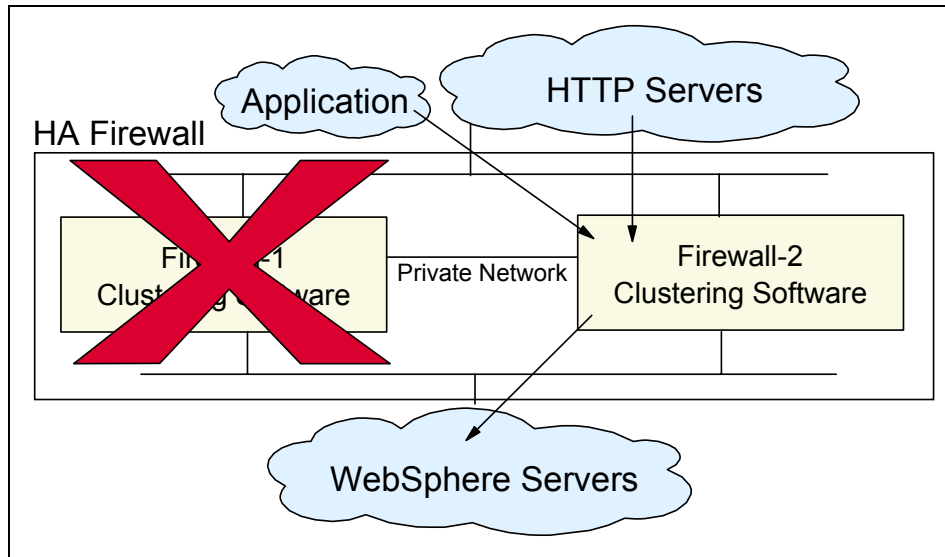


Figure 13-12 Clustered firewall after failover

### 13.3.2 Using a network sprayer

The HA firewall through clustering service discussed above is very costly and its configuration is complicated. It can be used by high-end customers. You can also set up an HA firewall with a network sprayer such as WebSphere Edge Components' Load Balancer, as shown in Figure 13-13 on page 519. The Load Balancer (LB) is a load balancing product that divides up the workload generated by new connections among a group of back-end servers. This can be done either by changing the assignment between the host name and the IP address (DNS redirection), or by rerouting TCP and UDP traffic directly to the server with the lowest workload. LB also recognizes server failures and automatically keeps new requests from being dispatched to the failed server.

There are two approaches to using the WebSphere Edge Components' sprayer to build an HA firewall environment:

- ▶ Interactive Session Support (ISS)
- ▶ Load Balancer

## Interactive Session Support

The ISS is responsible for the load balancing in conjunction with a Domain Name System (DNS) server. The load balancing is achieved either with an intelligent round-robin mechanism or by recalculating the load of the back-end servers using several system parameters, such as CPU load or memory utilization. The name server keeps a DNS entry representing a group of available servers. ISS constantly monitors the workload of the back-end servers and periodically replaces the IP address the entry points to with the IP address of the server that has the lowest workload at that time. This approach works fine with TCP/IP connections that are open for a few minutes, but it will pose problems with very short connections, such as requests to static HTTP pages. After LB starts routing requests to another server, the previous server will soon have completed all running requests and remain idle until LB starts to recalculate the next routing path. This approach will not be very efficient in regard to load balancing, but is easy to implement. You do not need a dedicated server running as an ISS monitor, because the ISS monitor runs directly on one of the back-end servers. Also, if the master ISS server fails, one of the remaining servers is chosen as the new ISS master server, thus making the system automatically highly available, as long as the DNS server does not fail.

The problem is that application clients and other DNS servers cache the DNS entries returned by ISS. By default, these entries are cached for 24 hours, depending on the time to live (TTL) information for the DNS entry. The time to live could be changed to the average duration of connections, but using a very short time to live will increase the number of DNS requests to your server dramatically, thus increasing the load and Internet traffic on your servers and networks. Therefore, if you want to keep the load of your DNS server at a reasonable level, changes have to be propagated in fifteen to thirty minute intervals, which is not adequate for highly available environments.

Using the name server module shipped with ISS, you can build up a DNS server of your own to serve only ISS requests. However, your DNS server will be a single point of failure, because you will not be able to use the other existing DNS servers as your secondaries. This is because the ISS DNS server updates its zone information very frequently. It is not practical to use secondary DNS servers, as the information on them will not always be synchronized with that on the primary ISS name server. As discussed so far, using ISS to make firewalls

highly available is not the best choice, because of the drawbacks mentioned and because you hardly get the takeover time demanded for high availability. Nevertheless, ISS provides an easy way to achieve simple load balancing among application servers.

### **Load Balancer function**

A more sophisticated load balancing tool is the Load Balancer. In contrast to the ISS functionality, the clients never get to see the IP addresses of the real application server they use. They have to target a cluster IP address configured on the LB server. The requests are then rerouted by the Load Balancer to the server with the lowest workload. The LB recalculates the workload of the servers either with information collected by the LB itself, such as the number of active connections and new connections, or with system information collected by the ISS software running locally on the servers, such as CPU load or memory utilization.

Since the workload is recalculated for every new connection, the connections always go to the least-loaded server. This can potentially cause problems with applications and protocols that require multiple TCP/IP connections to the same server in order to complete a transaction. If such connections get dispatched to different servers, the transaction will fail if the servers are not sharing state information. The data returned by the server, which usually consumes more network bandwidth than the initial request (for example, the content of an HTTP page), is sent directly from the server to the client, and the network load on the Load Balancer remains at a reasonable level.

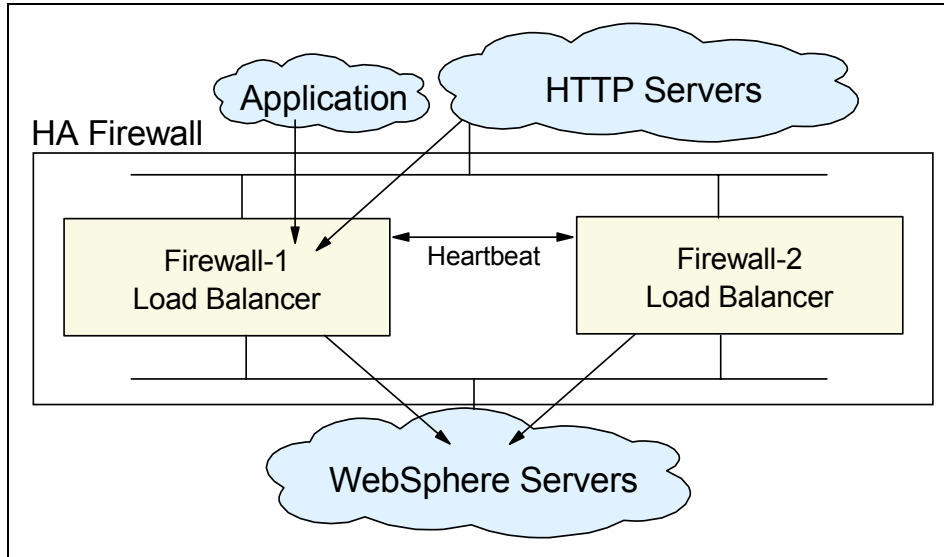


Figure 13-13 Load Balancer configured firewall

If you use ISS, high availability is already built in, because the monitor software runs on all application servers, and the server with the highest priority (set in its configuration file) is used to collect information about the workload from the other servers and reconfigure the DNS server. On the other hand, if you use the Load Balancer function, the LB server is a single point of failure in the system. To prevent this, LB gives us the option to configure a backup Load Balancer server that automatically takes over in case of a failure. The actual load information and client routing tables are shared between the two LB servers, so nearly all connections can be preserved in the case of a breakdown. Some external scripts are automatically executed during takeover, and they can be modified to provide the high availability of the firewall.

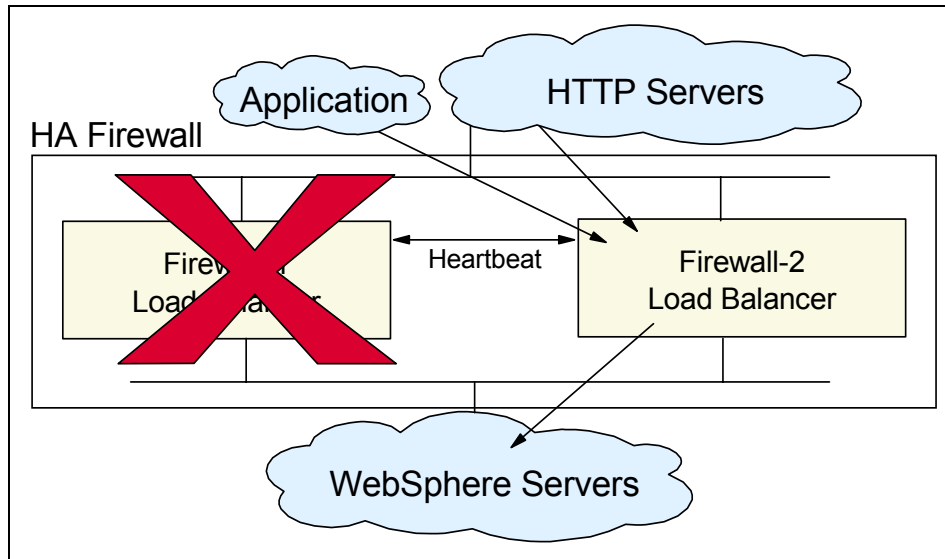


Figure 13-14 Load Balancer configured firewall after failover

### 13.3.3 Conclusions

In making a comparison of the high availability features of clustering and sprayer, the following aspects should be considered.

HACMP is a very complex product and needs to establish multiple connections between the two servers. These connections must be allowed by the firewall. If SSH is used for HACMP communication between the servers, we have the benefit of encryption and authentication. However, using SSH will cause some of the HACMP scripts to report errors in the cluster verification, which can be ignored. Configuration and troubleshooting of this solution is not an easy job and produces an environment that is not easy to understand. Therefore, the firewall administrator must have a good understanding of HACMP to keep this solution running.

Using the high availability function provided with LB is fairly simple. The Load Balancer executes shell scripts for startup and takeover events, which configure the cluster IP addresses either to the loopback device or to the network card, depending on the state of the system. These scripts can be customized for your environment and can include commands to start and stop application proxies and generate alerts for takeover events. Because LB only uses one TCP connection on a dedicated port for the heartbeat, and ping for testing network functionality, there are few changes on the firewall configuration, so setup is fairly easy.



Both software packages can test network connectivity to determine if a network card on the active firewall has failed or if there is a general network failure by pinging other systems on that network.

## 13.4 Network file system high availability

The network file system is a key part of any enterprise application. For WebSphere, you can use a network file system (NFS) to store your data, files, and various logs. Making this network file system highly available is the topic we will address here.

You can use clustering software such as HACMP, MC/ServiceGuard, or Sun Cluster to implement a highly available NFS server.

A file system that is exported by one node can be acquired by another node. We can design NFS cross mounts, where one cluster node mounts a file system through NFS on another cluster node. A cluster node that was originally an NFS client can become an NFS server after the failure of its partner node.

As shown in Figure 13-15 on page 522, we can use a similar approach to configure a clustering system for NFS, as discussed previously for the different clustering software products.

**Note:** For HACMP cluster systems, you should not use the standard `/etc/exports`. Instead, you should use `/usr/sbin/cluster/utlis/cl_export_fs` script.

In a network file system, the server system passes a file handle to the client system, which consists of a major device number, a minor number, an i-node, and a generation number. For the cluster system with shared disks, as shown in Figure 13-15 on page 522, all four items should be equal in each node that can take over these disks. After takeover, the file handle held by an NFS client should still be valid on the new server. When we create a logical volume, the minor number, i-node number, and generation number are the same for each node because they are self-contained within the logical volume. However, the default major device number may be different, since the logical volume inherits its major device number from the major device number of the volume group that contains it. The default major device number is assigned the lowest unused number. In a cluster system of multiple nodes, this next lowest unused major device number will not necessarily be the same. Note that major device numbers are assigned to devices such as TTYs, printers, disks, etc. Therefore, configure the volume group in the shared disks with the same major device number for all nodes in the cluster, so that your file handle is still valid after a failover.

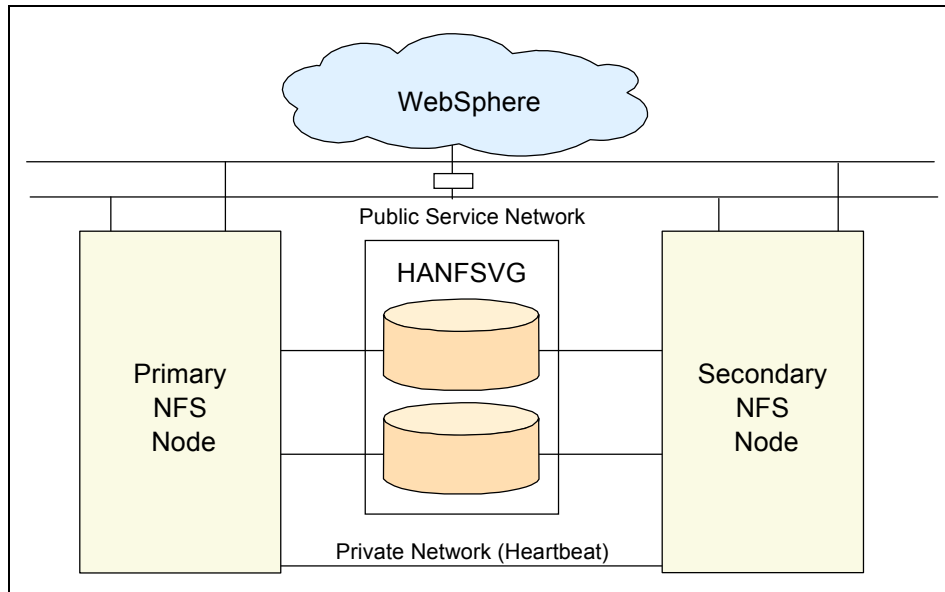


Figure 13-15 Highly available network file system for WebSphere

## 13.5 Summary

We have discussed various techniques to implement high availability end-to-end WebSphere production systems. Availability commonly refers to *uptime*. Using techniques discussed here, you can build the entire WebSphere system with 99.5% availability or better.

High availability is not only for 7x24 businesses. There are two types of downtimes: planned downtime and unplanned downtimes. The techniques we discussed above are for both planned downtime and unplanned downtime. Clustering techniques make rolling upgrades of hardware and software and hot replacement possible. Nobody knows when unplanned downtime will occur; it may occur during your business hours, even though you are in a 5x8 business. Clustering techniques help the system with automatic fault detection and service recovery. If you want to avoid interruptions to your operations due to system failures, then you need to use these techniques whether you are a 7x24, 6x20, or 5x8 business. Non-7x24 business hours provide opportunities for planned downtime for system maintenance and software/hardware upgrades off-line, but your system can still fail at any time due to hardware, software, and network problems.

High availability implementation needs high investment in hardware, software, and skilled personnel. Therefore, it is important to evaluate how much you will lose if your system is down during your operation hours.

WebSphere production system availability is determined by the weakest link in the WebSphere system chain. Therefore, it is very important to evaluate each part of the end-to-end WebSphere system high availability and eliminate all SPOFs, as shown in Figure 13-16.

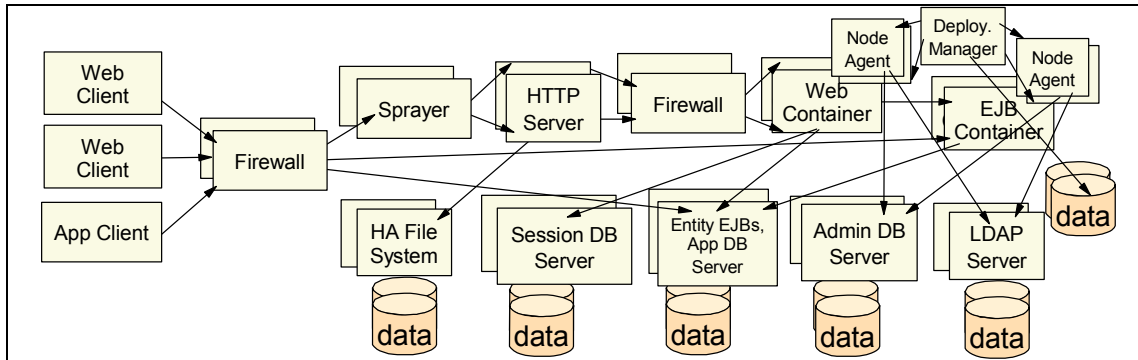


Figure 13-16 An end-to-end WebSphere system that removes SPOFs

There are two kinds of availability:

- Process availability
- Data availability

Process availability is achieved by multiple processes of an application, such as WebSphere workload management using server clusters, multiple Web servers, multiple database instance processes, multiple firewall processes, and multiple LDAP server processes. Usually, clients find the available process using 1-to-n mapping, redirection (IP spraying), or transparent IP takeover.

Data availability is achieved by replication or clustering. When data is shared by multiple processes, data integrity should be ensured by distributed lock management. Data is either stored in memory or on disk. For in-memory or local data, we need to maintain client request affinity to access the same data. It is very important to make sure that data inconsistencies be corrected before any process uses data, since a failed process may damage data integrity.

Depending on data change and access frequencies, we have different approaches to achieve data high availability:

► **Type I. Static data**

There are no changes for a period of months. An example is software install binaries. This static data is usually placed in individual hosts. For convenience of management, it can also be placed in shared disks or file systems.

► **Type II. Rarely changing data with planned change time (change period: several hours to several days)**

Examples are firewall configuration files, HTTP server configuration files, WebSphere configuration files, or HTTP static files. You can copy these files to different nodes (replication). However, an HA file system can help to minimize your administration burden. If, for example, you have 10 Web servers and you need to copy HTTP files to 10 Web servers every day (assuming that you change Web pages once a day), content management software could be used to reduce the administrative work involved in managing this data.

► **Type III. Rarely changing data with unplanned change time**

Examples are LDAP data and WebSphere administrative repository. Clustering or replication can be used for high availability, as we have discussed.

► **Type IV. Active data with frequent accesses and frequent changes**

Examples are Entity EJBs data, session data, and application data. We have discussed how to make these data and data management services highly available.

Data high availability is more difficult to implement than process high availability. Most importantly, data high availability and process high availability are both needed to complete the task. Data high availability is essential in most applications. It does not make any sense for a business to run a process if that process cannot access required data. For example, there is little value in running a stock brokerage system if stock data and trading session data are unavailable. It does not help to have the WebSphere EJB container process available if Entity EJBs cannot access their data states, or to have the WebSphere Web container process if servlets cannot access needed HTTP session states.

We have discussed aspects and techniques for building an end-to-end highly available WebSphere production system. Through these techniques, we can achieve both data high availability and process high availability for the WebSphere system.



## Part 5

# **Performance monitoring, tuning, and coding practices**





## Dynamic caching

Server-side caching techniques have long been used to improve Internet performance of Web applications. In general, caching improves response time and reduces system load. Until recently, caching has been limited to *static content*, which is content that rarely changes. However, performance improvements are greatest when *dynamic content* is also cached. Dynamic content is content that changes frequently, or data that is personalized. Caching dynamic content requires proactive and effective invalidation mechanisms, such as event-based invalidation, to ensure the freshness of the content. Implementing a cost effective caching technology for dynamic content is essential for the scalability of today's dynamic, data-intensive, e-business infrastructures.

This chapter covers the following topics:

- ▶ “Introduction” on page 528
- ▶ “Using WebSphere dynamic cache services” on page 534
- ▶ “WebSphere dynamic caching scenarios” on page 543
- ▶ “WebSphere external caching scenarios” on page 566
- ▶ “Conclusion” on page 583
- ▶ “Benchmarking Trade3” on page 585

## 14.1 Introduction

*Performance* became a term frequently used during every WebSphere Application Server project and caching became a key technology to reduce cost and improve the response time.

This chapter introduces the dynamic cache service provided by WebSphere Application Server V5.1. We explain how and when caching can be used to improve the performance of WebSphere Application Server solutions.

We are looking at caching from an administrative rather than development point of view. This means that the application can benefit from a cache without requiring modification. In most cases, enabling caching is a task performed during application deployment, though in some cases the application developers need to perform specific customization, which are required by the caching framework. For example, when it comes to more granular caching techniques, such as command caching, application developers have to develop the code according to WebSphere caching specifications. We address such customization in “Command result caching” on page 550.

### 14.1.1 WWW caching services

First, there are two basic types of content from the caching point of view:

- Static content

Static content does not change over long periods of time. The content type can be HTML, JSP rendering less dynamic data, GIF, JPG, etc.

**Note:** JSP rendering less dynamic data is considered to be static content from the caching point of view. When JSP is rendered, static HTML code is generated and this is cached.

- Dynamic content

Dynamic content includes frequently updated content (such as exchange rates), as well as personalized and customized content. Although it is changing content, at the same time it must be stable over a long enough time for meaningful reuse to occur. However, if some content is very frequently accessed, such as requests for pricing information of a popular stock, then even a short time of stability may be long enough to benefit from caching.

Secondly, it is important to understand that caching goes beyond an applications' own infrastructure. For example, there are many cache instances currently



deployed on the Internet. Figure 14-1 outlines the different tiers of caching for Web applications.

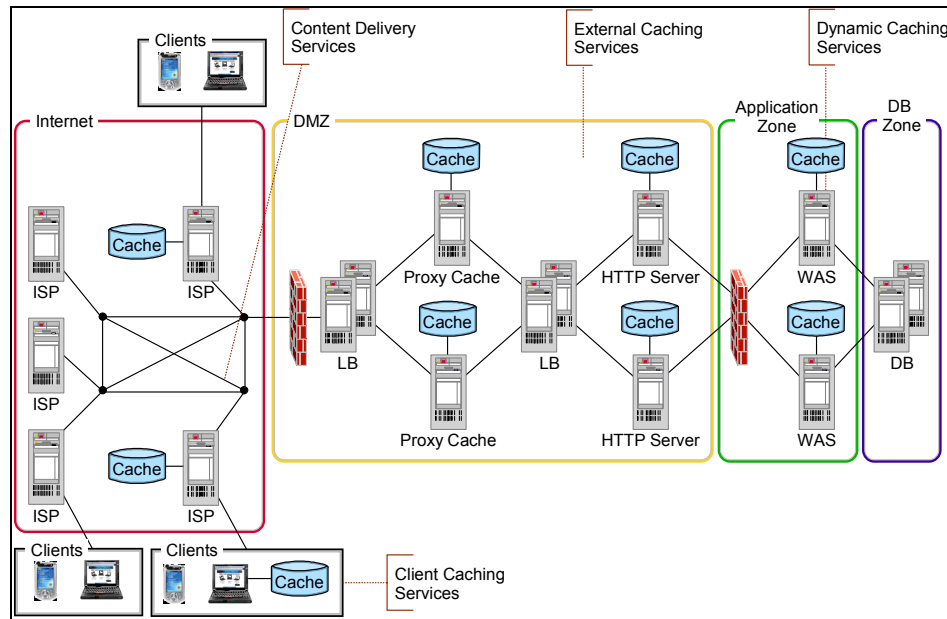


Figure 14-1 Content caching on the internet

The content caching tiers are:

- Client caching services

These services can be provided by any kind of Web browsers. Web browsers can store received content locally on the client device. Unfortunately, very often you will experience that interpretations of HTTP cache directives are browser dependent.

- Internet content delivery services

These services provide the replication and caching infrastructure on the Internet. The infrastructure is based on standard concepts, protocols and taxonomy. The main actors are Internet Services Providers (ISP) and Content Delivery Network (CDN) service providers, such as Akamai (<http://www.akamai.com>), or Speedera (<http://www.speedera.com>).

The physical infrastructure is built using cache appliances such as Network Appliance (<http://www.netapp.com>), Blue Coat (<http://www.bluecoat.com>), Cisco Cache Engine (<http://www.cisco.com>), or InfoLibria DynaCache (<http://www.infolibria.com>).

These services can also support the dynamic composition of page fragments in the network using a standard called Edge Side Include (ESI, <http://www.esi.org>). “WebSphere external caching scenarios” on page 566 explains how WebSphere Application Server V5.1 supports ESI.

► External caching services

These services provide front-end caching of both static and dynamic pages. The technical component can be the Caching Proxy, part of WebSphere Edge Components, IBM HTTP Server, or the Web server plug-in. All of them can intercept requests from a client, retrieve the requested information from content-hosting machines, and deliver that information back to the client. In addition, they can store cacheable content in a local cache (memory and/or disk) before delivering it to the requestor. This enables the external caches to satisfy subsequent requests for the same content by delivering it directly from the local cache, which is much quicker than retrieving it again from the content host.

These services provide both whole page and fragment caching; and again the dynamic composition of page fragments is based on ESI.

We further discuss external caching services in “WebSphere external caching scenarios” on page 566.

► Dynamic caching services

These services provide the caching of servlets, JSPs, commands, Web services, or EJBs. They work within an application servers’ Java Virtual Machine (JVM) by intercepting calls to cacheable objects.

“WebSphere dynamic caching scenarios” on page 543 focuses on these dynamic caching services.

This chapter is focused on the dynamic caching services and their integration with external cache services.

In most cases only a combination of dynamic and external services can power high volume Web sites to achieve the required level of scalability and performance. When implementing such a combination, the impact on data integrity and accuracy needs to be also considered. For example, if you have a WebSphere Commerce implementation and you use the WebSphere Edge Components to cache product pages, it can take some time until the WebSphere Edge Components cache is synchronized with updates to the product database used by WebSphere Commerce. This time can be as long as the time-to-live parameter set for the product pages.

The question to be asked is, how can you make sure that all these different caching services will support your specific application? The answer to this question is to use standard protocols, such as HTTP/1.1 and ESI/1.0.

Traditional HTTP caching is mostly used in cases where the whole page is cacheable and does not contain personalized content. On the other hand, ESI allows caching to operate on the level of fragments rather than whole pages.

HTTP/1.1 provides basic cache mechanisms in the form of implicit directives to caches. It also allows the use of explicit directives for the HTTP cache called "Cache - Control". The Cache-Control header allows a client or server to transmit a variety of directives in either requests or responses. These directives typically override the default caching algorithms.

Detailed information on the HTTP/1.1 caching directives are included in RFC 2616 (<http://www.faqs.org>). Additionally, RFC 3143 lists known HTTP Proxy Caching problems.

**Note:** An additional level of caching, which is not described in this chapter, is database caching. There are a number of products available for database caching, for example DB2 DBCache, TimesTen, Oracle 9i IAS - Relational Cache, Versant's enJin.

For more information on database caching, refer to this Web site:

<http://www.almaden.ibm.com/u/mohan/>

Here you find a link to the presentation "Caching Technologies for Web Applications".

### 14.1.2 Fragment caching

Most dynamic Web pages consist of multiple smaller and simpler page fragments. Some fragments are static (such as headers and footers), while others are dynamic (such as fragments containing stock quotes or sport scores). Breaking a page into fragments or components makes effective caching possible for any page, even a highly dynamic page.

These are the main goals of fragment caching:

- ▶ Achieve benefits of caching for personalized pages.
- ▶ Reduce cache storage requirements, by sharing common fragments among multiple pages which helps maximize fragment reusability.
- ▶ Move cache into the network to multiply above benefits.

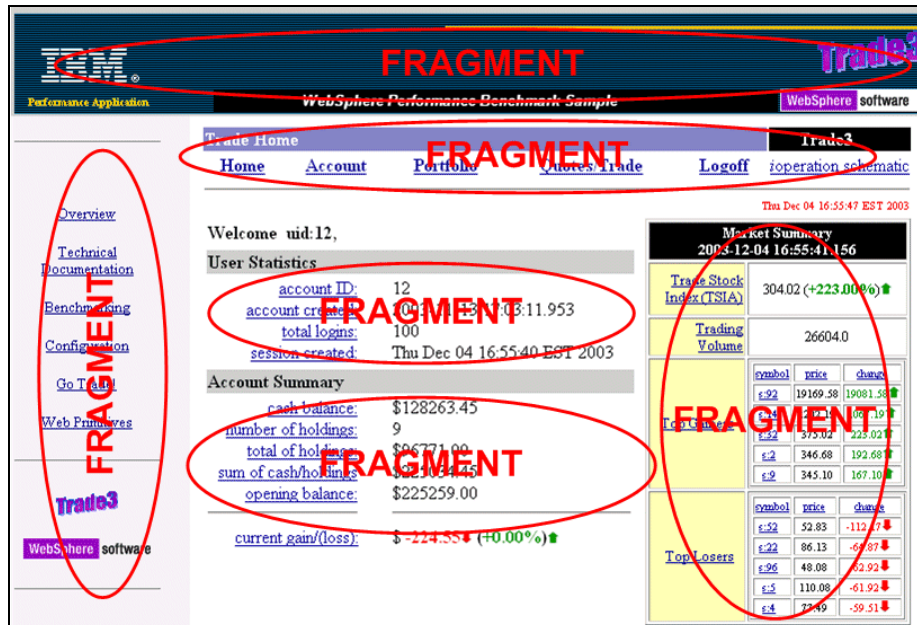


Figure 14-2 Page fragments

Figure 14-2 shows that when a page is broken down into fragments based on reusability and cacheability, some or all of the fragments (for example, headers, footers, and navigation bars for all users; market summary for user groups) may become reusable and cacheable for a larger audience. Only fragments that are not cacheable need to be fetched from the back-end, thereby reducing server-side workload and improving performance. Even a very short time of caching can improve performance, for example if you can cache the Market Summary information just for 10 seconds, it can make a big difference during peak hours.

Web pages should be fragmented to cache dynamic content effectively within the enterprise infrastructure and at the content distribution network. However, in some cases, even caching the most granular, final formatted fragment is not sufficient. Under such circumstances, caching at the raw data level is the next granular technique that can be used (see 14.3.2, “Command result caching” on page 550).

Web page design also plays an important role in determining where dynamic data is cached. One example is personalized pages. Although personalized, these pages contain user specific, nonuser-specific, locale sensitive, secure, non security sensitive dynamic data. To maximize the benefit of caching dynamic content, these types of pages should be fragmented as finely as possible. They

can be cached independently at different locations. For example, the nonuser-specific, non security sensitive fragments or components are generally useful to many users, and thus can be cached in a more public space and closer to the users. The security sensitive data should be cached behind the enterprise firewall, yet as close to the edge of the enterprise as possible.

### 14.1.3 Dynamic caching scenarios

The key issue with caching dynamic content is to determine what should be cached, where caching should take place, and how to invalidate cached data.

In a multi-tier e-business environment, the WebSphere Application Server dynamic cache service can be activated at the business logic and/or presentation layer. It can also control external caches on servers, such as WebSphere Caching Proxy or IBM HTTP Server. When external caching is enabled, the cache matches pages with their universal resource identifiers (URIs) and exports matching pages to the external cache. The contents can then be served from the external cache instead of the application server, which saves resources and improves performance.

To simulate real production systems, we explain the following scenarios:

- ▶ WebSphere dynamic caching
  - At the presentation logic layer (servlet/JSP resulting caching)
  - At the business logic layer (command result caching)
- ▶ WebSphere external caching
  - Using Web server plug-in with ESI
  - Using IBM HTTP Server with FRCA
  - Using WebSphere Caching Proxy

We do not focus directly on the caching techniques used by Content Delivery Networks, but we explain the IBM Web server plug-in, which has ESI surrogate capabilities.

When you design the caching solution for your application, you may want to combine different caching techniques. For example you may use a combination of dynamic caching and Web server plug-in caching, or a dynamic caching and WebSphere Caching Proxy combination. This chapter provides you with information that helps you to design your individual caching solution.

We are using the redbook's WebSphere Application Server infrastructure, which is described in detail in Chapter 7, "Implementing the sample topology" on page 255, for our scenarios. We use the WebSphere Performance Benchmark sample application Trade3, because apart from other significant features, Trade3 supports key WebSphere components such as dynamic caching.

As you can see in Figure 14-3 we can use caching services at different tiers of the application infrastructure. Our scenarios explain the appropriate caching techniques for the different tiers.

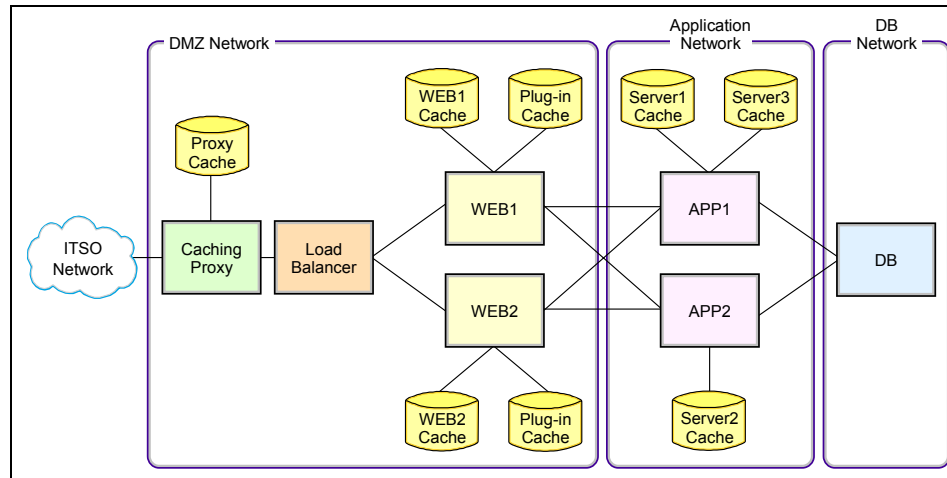


Figure 14-3 Redbook caching infrastructure

## 14.2 Using WebSphere dynamic cache services

Before we dive into the different scenarios, we explain the following general configuration tasks:

- ▶ Installing Dynamic Cache Monitor
- ▶ Enabling and configuring WebSphere dynamic cache service

### 14.2.1 Installing Dynamic Cache Monitor

The Dynamic Cache Monitor (Cache monitor) is an installable Web application that displays simple cache statistics, cache entries and cache policy information. The CacheMonitor.ear file is available in the <install\_root>/installableApps directory of your application server. For security reasons, you should not use the same host and port as used for your application. Therefore you first need to create virtual hosts and host aliases for the Cache monitor application:

1. Create a new virtual host called "cache\_monitor". Click **Environment -> Virtual Hosts -> New** in the Administrative Console. Enter **cache\_monitor** into the Name field and click **Apply**.

For vertical clustering, this must be repeated on every machine in the cluster.

2. Create a new host alias (verify which port numbers are available on your system first) for the cache\_monitor host:
  - a. Select **Host Aliases** from the Additional Properties pane. Click **New**.
  - b. Enter a \* (asterisk) into the Host Name field and add **<your\_port\_nbr>** (for example 9082).
3. Click **OK** and save your configuration.
4. Add a new HTTP transport port to your application server(s):
  - a. Click **Servers -> Application Servers -> <your\_app\_server> -> Web Container -> HTTP transports**. Click **New**.
  - b. Enter \* for the Host and **<your\_port\_nbr>** (for example, 9082) for the Port.
  - c. Click **OK** and save your configuration.
5. Repeat steps 2 to 4 for each application server in your cluster.

We are using three clustered application servers in our environment: Trade3Server1 and Trade3Server3 reside on the same machine, so we need different ports for these two application servers. Trade3Server2 resides on a different machine, so we can use the same port number as was used for Trade3Server1. Our configuration is as follows:

- Port 9082 for Trade3Server1 and Trade3Server2
- Port 9085 for Trade3Server3

Now that these preliminary tasks are done, you can install the Cache monitor application. Here is how to do this:

1. Open the Administrative Console and verify whether the Cache monitor is already installed. To do so, click **Applications -> Enterprise Applications**. See Figure 14-4 on page 536. If CacheMonitor is not displayed in this list, click **Install**.

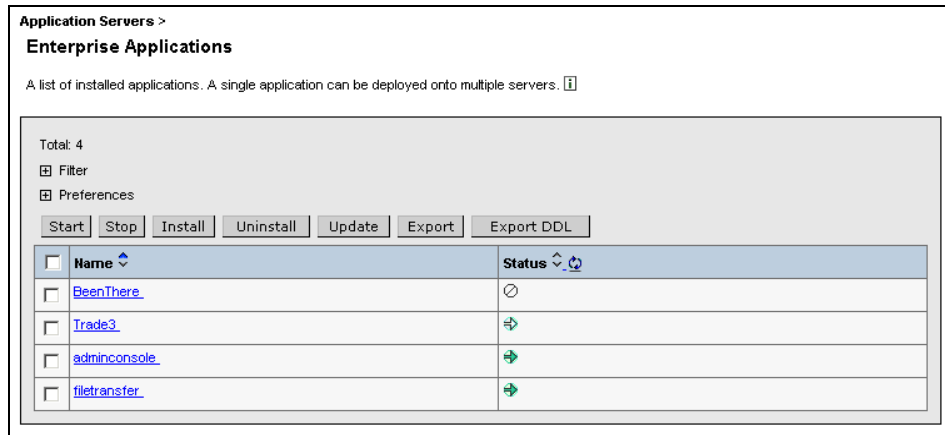


Figure 14-4 Installed Enterprise Applications

- On the Preparing for the application installation panel (Figure 14-5), browse to the <install\_root>/installableApps Server path, select the **CacheMonitor.ear** file. Click **OK**, then click **Next**.

#### Preparing for the application installation

Specify the EAR/WAR/JAR module to upload and install.

Path:

Browse the local machine or a remote server:

☐ Local path:  [Browse...](#)

☒ Server path:  [Browse...](#)

[i](#) Choose the local path if the ear resides on the same machine as the browser. Choose the server path if the ear resides on any of the nodes in your cell context.

Context Root: Used only for standalone Web modules (\*.war)

[i](#) You must specify a context root if the module being installed is a WAR module.

[Next](#) [Cancel](#)

Figure 14-5 Cache monitor installation - Select CacheMonitor.ear

- On the next panel, enter **cache\_monitor** for the Virtual Host. Click **Next -> Continue**.



**Preparing for the application installation**

You can choose to generate default bindings and mappings. [i](#)

☐ Generate Default Bindings

Override:	<input checked="" type="radio"/> Do not override existing bindings <input type="radio"/> Override existing bindings	<a href="#">i</a> Generate default bindings for existing entries and over write them.
Virtual Host	<input type="radio"/> Do not default virtual host name for web modules <input checked="" type="radio"/> Default virtual host name for web modules: <input type="text" value="cache_monitor"/>	<a href="#">i</a> The virtual host to be used for this web module.
Specific bindings file:	<input type="text"/> <input type="button" value="Browse..."/>	<a href="#">i</a> Optional location of pre-defined bindings file.

Figure 14-6 Select Virtual Host - cache\_monitor

- Accept the default options on the next panel (Step 1) and click **Next** once again. See Figure 14-7.

→ **Step 1 : Provide options to perform the installation**

Specify the various options available to prepare and install your application.

AppDeployment Options	Enable
Pre-compile JSP	<input type="checkbox"/>
Directory to Install Application	<input type="text"/>
Distribute Application	<input checked="" type="checkbox"/>
Use Binary Configuration	<input type="checkbox"/>
Deploy EJBs	<input type="checkbox"/>
Application Name	<input type="text" value="Dynamic Cache Monitor"/>
Create MBeans for Resources	<input checked="" type="checkbox"/>
Enable Class Reloading	<input type="checkbox"/>
Reload Interval in Seconds	<input type="text" value="0"/>
Deploy WebServices	<input type="checkbox"/>

Figure 14-7 Cache monitor installation - Installation options

5. Step 2 - Map virtual hosts for Web modules (Figure 14-8).

We have created a virtual host called “cache\_monitor” in step 1 on page 534 which we are now referring to.

Check **Dynamic Cache Monitor** and select the **cache\_monitor** Virtual Host from the drop-down menu. Then click **Next**.

→ Step 2: Map virtual hosts for web modules

Specify the virtual host where you want to install the Web modules contained in your application. Web modules can be installed on the same virtual host or dispersed among several hosts.

☐ Apply Multiple Mappings

<input type="checkbox"/> Web Module	Virtual Host
<input checked="" type="checkbox"/> Dynamic Cache Monitor	cache_monitor

Previous

Next

Cancel

Figure 14-8 Cache monitor installation - Map virtual hosts for Web modules

6. Step 3 - Map modules to application servers (Figure 14-9).

On this panel, you need to select your application server and the **Dynamic Cache Monitor** Module. Click **Apply**, then click **Next**.

→ Step 3: Map modules to application servers

Specify the application server where you want to install modules contained in your application. Modules can be installed on the same server or dispersed among several servers.

Clusters and Servers:

WebSphere:cell=dmNetwork,cluster=MyEJBCluster  
WebSphere:cell=dmNetwork,cluster=MyWebCluster  
WebSphere:cell=dmNetwork,cluster=Trade3Cluster

Apply

<input type="checkbox"/> Module	URI	Server
<input checked="" type="checkbox"/> Dynamic Cache Monitor	CacheMonitor.war,WEB-INF/web.xml	WebSphere:cell=dmNetwork,node=dmManager,cluster=MyEJBCluster

Previous

Next

Cancel

Figure 14-9 Cache monitor installation - Map modules to application servers

7. Step 4 - Map security roles to users/group (Figure 14-10 on page 539).

Select **Administrator** and **All Authenticated**, then click **Next** once more.

→ **Step 4: Map security roles to users/groups**

Each role defined in the application or module must be mapped to a user or group from the domain's user registry.

Lookup users    Lookup groups

<input type="checkbox"/> Role	Everyone?	All Authenticated?	Mapped Users	Mapped Groups
<input checked="" type="checkbox"/> Administrator	<input type="checkbox"/>	<input checked="" type="checkbox"/>		

Previous    Next    Cancel

Figure 14-10 Cache monitor installation - Map security roles to users/groups

- Confirm the installation on the Summary window as shown in Figure 14-11, click **Finish** and **Save** your changes to the master configuration.

→ **Step 5: Summary**

Summary of Install Options

Options	Values
Distribute Application	Yes
Use Binary Configuration	No
Cell/Node/Server	<a href="#">Click here</a>
Enable Class Reloading	No
Create MBeans for Resources	Yes
Deploy EJBs	No
Reload Interval in Seconds	0
Application Name:	Dynamic Cache Monitor
Directory to Install Application	
Pre-compile JSP	No
Application Name	Dynamic Cache Monitor
Deploy WebServices	No

Previous    Finish    Cancel

Figure 14-11 Cache monitor installation - Summary

- Regenerate plugin-cfg.xml. Click **Environment -> Update Web Server Plugin -> OK**.
- Restart your application server(s).

Generally, you can access the Cache monitor using a Web browser and the URL

`http://<your_hostname>:<your_port_number>/cachemonitor`

We are using these URLs:

- ▶ `http://app1.itso.ibm.com:9082/cachemonitor/` for Trade3Server1
- ▶ `http://app2.itso.ibm.com:9082/cachemonitor/` for Trade3Server2
- ▶ `http://app1.itso.ibm.com:9085/cachemonitor/` for Trade3Server3

If your application server is configured for global security, you need to provide a valid user ID and password after invoking the above URL.

Figure 14-12 shows the Cache monitor start page which allows access to the cache statistics and some configurable parameters. These are configured using the Dynamic Cache Administrative Console (see “Enabling dynamic cache service” on page 540).

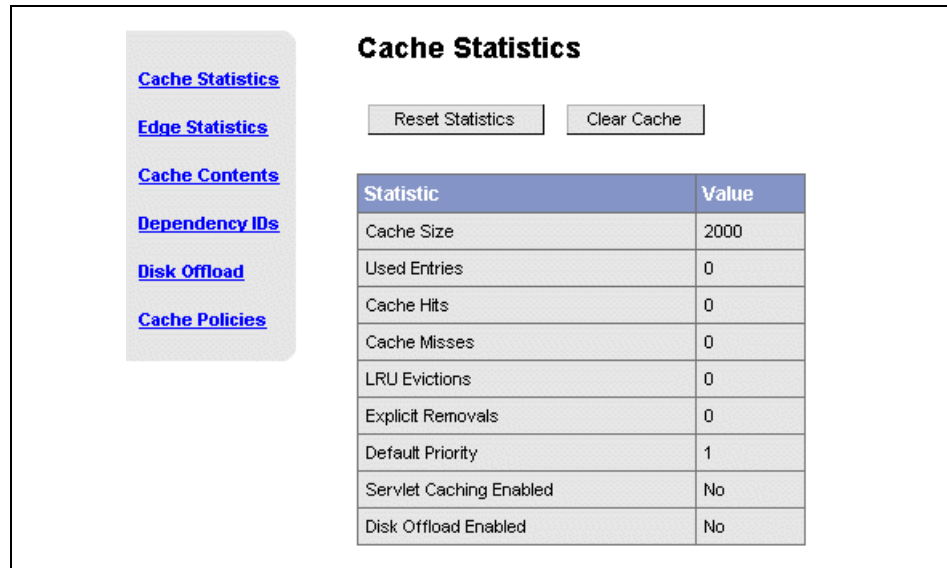


Figure 14-12 WebSphere Cache monitor start page

We explain some functionality of the Cache monitor as we work on our sample scenarios, but please use the WebSphere InfoCenter to fully understand all functionality. The WebSphere Information Center is available at:

<http://www.ibm.com/software/webservers/appserv/infocenter.html>

The ESI processor's cache is also monitored through the Dynamic Cache Monitor application. In order for the ESI processor's cache to be visible in the Cache monitor, the DynaCacheEsi application must be installed and the `esiInvalidationMonitor` property must be set to true in the `plugin-cfg.xml` file. See “External caching by Web server plug-in” on page 570 for more information on the ESI processor.

## 14.2.2 Enabling dynamic cache service

The dynamic cache service is an in-memory cache system that has disk off-load capability. Its caching behavior is defined in an XML file named `cachespec.xml`.

The cache policy is configured when building the XML configuration files. A graphical user interface (GUI) tool for building the cache policy files is available at

<http://www.alphaworks.ibm.com/tech/cachepolicyeditor>

for use with WebSphere V5.1.0 Service Pack 3 and above.

Unique ID strings distinguish unique entries in WebSphere Application Server's dynamic content caching solution. These ID strings can be defined declaratively with the XML configuration file or programmatically by the user's Java program. Please refer to the WebSphere InfoCenter for more information on this.

WebSphere Application Server dynamic cache service can control external caches. Different groups of external caches can be defined, each with its own set of member caches. The interface between WebSphere Application Server and the external cache is the *External Cache Adapter* provided by WebSphere Application Server. We elaborate more on this topic in "WebSphere external caching scenarios" on page 566.

The dynamic cache service includes an alternative feature named disk off-load, which stores the overflow cache entries on disk for potential future access. This feature removes the dynamic cache memory constraints experienced in previous WebSphere Application Server versions.

To enable dynamic caching for an application server, follow these steps:

1. Open the Administrative Console.
2. Click **Servers** -> **Application Servers** in the navigation tree.
3. Select your application server.
4. Select **Dynamic Cache Service** from the Additional Properties.
5. Select **Enable service at server startup** in the Startup state field.
6. Click **Apply** or **OK**.

All changes made to the dynamic cache service properties take effect after restarting the server.

Figure 14-13 on page 542 shows the configuration panel of the dynamic cache service in the Administrative Console.

Configuration		
<b>General Properties</b>		
Startup state:	<input checked="" type="checkbox"/> Enable service at server startup	<i>i</i> Click the check box to enable the dynamic cache service at the time of server startup.
Cache size	* 2000 entries	<i>i</i> A positive integer defining the maximum number of entries the cache will hold.
Default Priority	* 1	<i>i</i> The default priority for cache entries, determining how long an entry will stay in a full cache.
Disk offload:	<input type="checkbox"/> Enable disk offload Offload location: <input type="text"/>	<i>i</i> By default, the dynamic cache only maintains, at most, the number of entries configured in memory. If new entries are created while the cache is full, the priorities configured for each cache entry, along with a least recently used algorithm, are used to remove entries from the cache. As an alternative, disk offload can be configured and rather than removing the entries from memory, they will be copied onto the file system (the location is configurable) and can be accessed later.
Cache replication:	<input type="checkbox"/> <a href="#">Enable cache replication</a>	<i>i</i> Enable cache replication here.
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>		
<b>Additional Properties</b>		
<a href="#">External Cache Groups</a> Define sets of external caches controlled by WebSphere Application Server.		

Figure 14-13 Enabling dynamic caching in Administrative Console

As was mentioned before, the WebSphere Application Server uses JVM memory to store cached objects. Therefore it is important to know how much memory can be allocated for the cache and based on this information you can set the cache size to the proper value. You need to be able to estimate the total size of objects which can be cached during peak hours. For example, you could use Page Detailer to analyze the size of generated pages. Also, your application designer needs to understand the performance implication of heavy pages.

If the estimated total size of all cached objects is bigger than the available memory, you can enable the disk off-load option. You then need to configure priorities for the cached objects in the cachespec.xml file. Priority weighting is used in conjunction with the least recently used (LRU) algorithm. Based on this algorithm it is decided which entries are moved from memory to disk if the cache runs out of storage space.

If you decide to use the disk off load option, you need to also configure your file system for fast I/O access. Depending on your hardware and software you can use various disk striping or caching techniques.

An additional feature of the dynamic cache services is *cache replication*. You can enable cache replication using the same configuration panel. Cache replication leverages the WebSphere internal replication service (DRS) that is also used for HTTP session memory-to-memory replication. A replication domain with at least one replicator entry needs to exist in order to replicate the data. The dynamic cache, in essence, connects to the replicator. We discuss cache replication further in 14.3.3, “Cache replication” on page 555.

## 14.3 WebSphere dynamic caching scenarios

These are the objectives for our dynamic caching scenarios:

- ▶ Describe and test servlet/JSP caching (see 14.3.1, “Servlet/JSP result caching” on page 544).
- ▶ Configure and test command result caching (see 14.3.2, “Command result caching” on page 550).
- ▶ Configure and test cache replication (see 14.3.3, “Cache replication” on page 555).
- ▶ Explore cache object invalidation (see 14.3.4, “Cache invalidation” on page 563).
- ▶ Explain troubleshooting steps (see 14.3.5, “Troubleshooting the dynamic cache service” on page 564).

We explore caching only in the application tier in our dynamic caching scenarios as shown in Figure 14-14 on page 544.

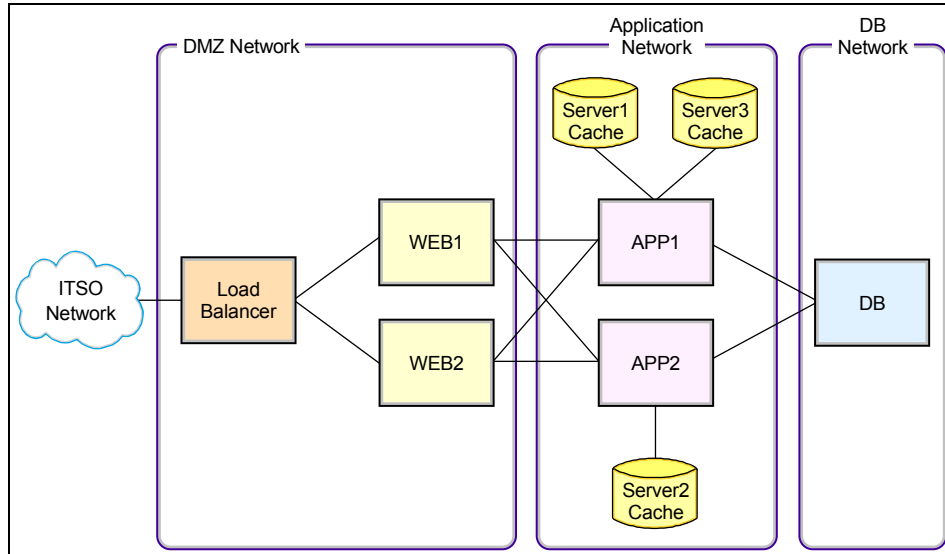


Figure 14-14 Redbook sample dynamic caching infrastructure

### 14.3.1 Servlet/JSP result caching

Servlet/JSP result caching intercepts calls to a servlet's service method, and checks whether the invocation can be served from cache. If the request cannot be served from cache, the servlet is invoked to generate the output that will be cached. The resulting cache entry contains the output and/or the side effects of the invocation, like calls to other servlets or JSP files. Figure 14-15 on page 545 shows all steps if the output is not cached. If the output is cached, then steps 6, 7 and 8 are skipped.



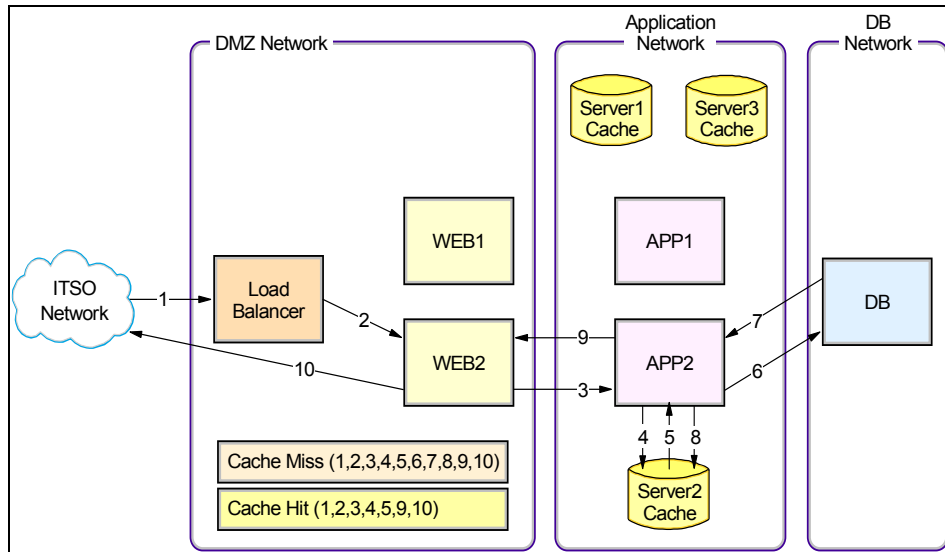


Figure 14-15 Servlet/JSP result caching - infrastructure view

Figure 14-16 gives a closer granularity look on steps 4 and 5. You can see that consequent requests are served only by the Web container, without involving the EJB container (steps 5 to 9).

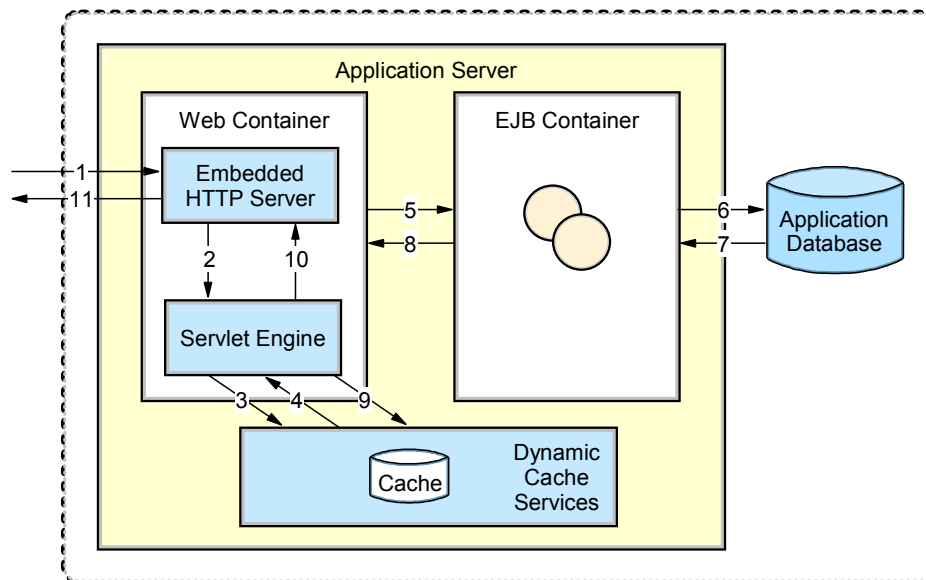


Figure 14-16 Servlet/JSP result caching - application server view

Servlet/JSP result caching can be based on:

- ▶ Request parameters and attributes
- ▶ The URI used to invoke the servlet or JSP
- ▶ Session information

Servlet/JSP result caching can be used to cache both whole pages or fragments. This concept is explained in Figure 14-17.

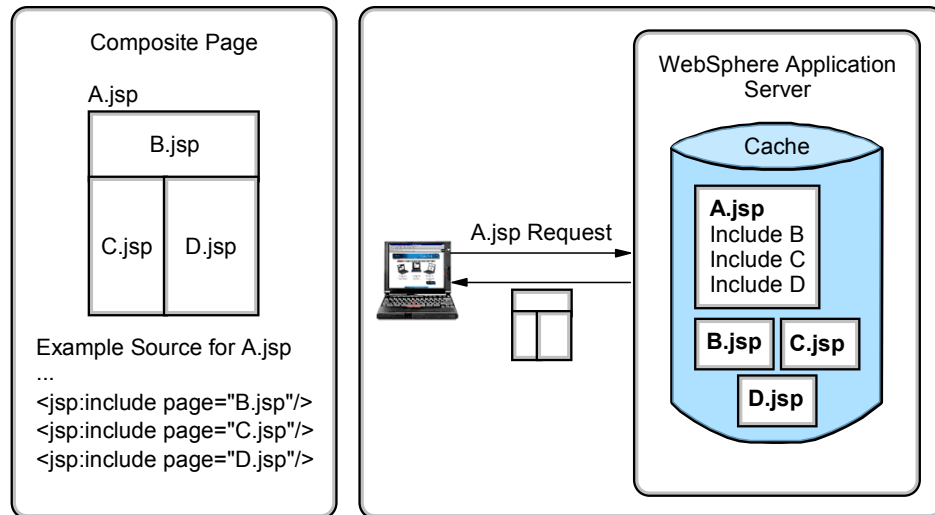


Figure 14-17 Servlet/JSP result caching - fragmentation

## Configuration steps

1. Enabling servlet and JSP result caching.

To enable servlet caching, you must enable the dynamic cache service for the Web container by performing these steps:

- a. Open the Administrative Console.
- b. Click **Servers - > Application Servers** in the console navigation tree.
- c. Select your application server.
- d. Click **Web Container**.
- e. Select the **Enable servlet caching** check box under the Configuration tab.
- f. Click **Apply** or **OK**.

2. Configure cachespec.xml.

In the runtime environment, the cache parses the cachespec.xml file on startup, and extracts from each `<cache-entry>` element a set of configuration

parameters. Every time a new servlet or other cacheable object initializes, the cache attempts to match each of the different <cache-entry> elements, to find the configuration information for that object. Different cacheable objects have different <class> elements. You can define the specific object a cache policy refers to using the <name> element.

As mentioned before, we are using Trade3 for this scenario. The cachespec.xml file for Trade3 can be found inside the WEB-INF directory of the Web module found in

```
<install_root>\installedApps\dmNetwork\Trade3.ear\trade3Web.war\WEB-INF
```

Refer to the WebSphere InfoCenter for a detailed description of cachespec.xml file elements. The InfoCenter can be found at

<http://www.ibm.com/software/webservers/appserv/infocenter.html>

### ***Editing Trade3 cachespec.xml file***

This section explains the cache entries for the Trade3 home servlet (see Figure 14-3 on page 534). This page contains the following cacheable fragments (formatted or non-formatted):

- ▶ Home servlet - Uses tradehome.jsp to format an output
- ▶ marketSummary.jsp - tradehome.jsp includes marketSummary.jsp
- ▶ Market summary command
- ▶ Account command
- ▶ Closed order command
- ▶ Holdings command

For the purpose of demonstrating servlet and JSP result caching we use the cachespec.xml file only with home servlet and market summary JSP cache entries.

1. First, the cache-entry element for the home servlet needs to be specified. In this case, the servlet's URI is "/app" so this will be our cache-entry's name element. Also, since this cache-entry is a servlet, the cache-entry class is "servlet". See Example 14-1:

#### *Example 14-1 Cache entry for home servlet*

---

```
<cache>
  <cache-entry>
    <class>servlet</class>
    <name>/app</name>
  </cache-entry>
</cache>
```

---

2. Example 14-2 on page 548 shows a definition of cache ID generation rules. The home servlet can be cached only when "action=home" and the servlet engine can retrieve the "JSESSIONID" cookie value (a user needs to be

logged in). Therefore one component of the cache ID will be the parameter "action" when the value equals "home". The second parameter will be JSESSIONID which equals to the valid user session ID. The URI for this servlet is "/app?action=home".

*Example 14-2 Cache ID for home servlet*

---

```
<cache-id>
  <component id="action" type="parameter">
    <value>home</value>
    <required>true</required>
  </component>
  <component id="JSESSIONID" type="cookie">
    <required>true</required>
  </component>
</cache-id>
```

---

3. Two dependency IDs are specified for the home servlet: Account\_UserID and Holdings\_UserID. Both IDs are used to identify user accounts by the UserID. The home servlet is session dependent and Account\_UserID is used to invalidate the home servlet when either LoginCommand or LogoutCommand are invoked. Additionally, Holdings\_UserID is used by OrderCompletedCommand to invalidate the home servlet upon the order completion. This is shown in Example 14-3. See also "Cache replication testing" on page 563.

*Example 14-3 Dependency IDs for home servlet*

---

```
<dependency-id>Account_UserID
  <component id="action" type="parameter" ignore-value="true">
    <value>home</value>
    <required>true</required>
  </component>
  <component id="uidBean" type="session">
    <required>true</required>
  </component>
</dependency-id>
<dependency-id>Holdings_UserID
  <component id="action" type="parameter" ignore-value="true">
    <value>home</value>
    <required>true</required>
  </component>
  <component id="uidBean" type="session">
    <required>true</required>
  </component>
</dependency-id>
```

---

4. A new cache entry is defined for the marketSummary.jsp, which is included by tradehome.jsp, which formats output from the home servlet command. The marketSummary.jsp can be invalidated based on time value (time-driven) or based on the update of data, which are rendered by marketSummary.jsp (event-driven). The ResetTradeCommand invalidates marketSummary.jsp using MarketSummary dependency ID.

*Example 14-4 Cache entry for marketSummary.jsp*

---

```
<cache-entry>
<class>servlet</class>
  <name>/marketSummary.jsp</name>
  <cache-id>
    <priority>3</priority>
    <timeout>180</timeout>
  </cache-id>
  <dependency-id>MarketSummary
  </dependency-id>
</cache-entry>
```

---

## Testing home servlet and market summary JSP

1. Log into the Trade3 application, for example using  
`http://apl.itso.ibm.com:9081/trade`
2. Open the Cache monitor using  
`http://apl.itso.ibm.com:9082/cachemonitor`
3. Clear the cache by clicking the **Clear Cache** button.
4. Click the Home link in the Trade3 application and use the Cache monitor to verify that both home servlet and market summary JSP are cached. If caching works properly, you can see that two cache entries are used (Figure 14-18 on page 550).

Statistic	Value
Cache Size	100
Used Entries	2
Cache Hits	0
Cache Misses	0
LRU Evictions	0
Explicit Removals	0
Default Priority	1
Servlet Caching Enabled	Yes
Disk Offload Enabled	No

Figure 14-18 Servlet/JSP result caching statistics

Home servlet and market summary JSP are in the cache as shown in Figure 14-19.

Template	Cache ID	Timeout (seconds)	Dependency IDs
<a href="#">/trade/marketSummary.jsp</a>	<a href="#">/trade/app.action=home;JSESSIONID=000140s0s7fkwhSqAnkIEgH5ZPtY3avd613;ISO-8859-1</a>	0	<a href="#">MarketSummary_Account.1</a>
<a href="#">/trade/marketSummary.jsp</a>	<a href="#">/trade/marketSummary.jsp;ISO-8859-1</a>	180	<a href="#">MarketSummary</a>

Figure 14-19 Servlet/JSP result caching statistics - Detailed view

All subsequent requests for these two entries will produce a cache hit as long as the objects are valid.

## 14.3.2 Command result caching

The Command Cache introduces the next level of granularity to dynamic content caching. Its primary goal is to serve the object's content from cache and thus minimize the execution of remote messages, such as back-end database JDBC calls, or calls to access data at remote non-database servers.

Generally, the Command Cache is used to cache dynamic data that requires back-end data retrieval or additional computation or manipulation. The command result cache forms a good synergy with the servlet/JSP result cache, because, in some cases, even caching the most granular, final formatted fragment is not sufficient. For example, a home page (Figure 14-2 on page 532) consists of two sets of information: “Account summary” that is highly personalized, and “Market summary” that is generalized information and usable by many users. “Account summary” is highly user sensitive and stored at the back-end server. In this case, it is not effective to cache the final formatted fragment, but rather use command

result caching.

The Command Cache is caching at the Java application level. To use Command Cache, user applications need to use the caching capabilities of the *WebSphere Command Framework API*. The WebSphere Command Framework is based on the *Command Design Pattern* widely used in Java programming paradigms. Typically, these applications use “setters” to set the command’s input states, one or more execute methods, and “getters” to retrieve the results. The results can be either formatted or raw data.

The Command Cache intercepts the execute method call of a command written for Command Cache and checks whether the command object can be served from the cache. If the command object does not exist in the cache, the logic in the execute method is performed and the resulting object is cached.

The caching behavior of Command Cache is defined declaratively with the XML cache policy file, which describes whether and how a command should be cached. Command Cache can be used at the presentation and/or business logic layer in multi-tier environments.

The Command Cache is easy to use. For existing applications that follow a *Model, View, and Controller (MVC)* design pattern, Command Cache can be implemented with minimal impact to existing code.

## Enabling command result caching

Cacheable commands are stored in the cache for re-use with a similar mechanism as used for servlets and Java Server Pages (JSP) files. However, in this case, the unique cache IDs are generated based on methods and fields present in the command as input parameters.

1. Developers need to implement the following steps during commands development:

- a. Create a command and define an interface.

The Command interface specifies the most basic aspects of a command.

You must define the interface that extends one or more of the interfaces in the command package. The command package consists of three interfaces:

- TargetableCommand
- CompensableCommand
- CacheableCommand

In practice, most commands implement the `TargetableCommand` interface, which allows the command to execute remotely. The code structure of a command interface for a targetable command is shown in Example 14-5:

#### Example 14-5 TargetableCommand interface

---

```
...
import com.ibm.websphere.command.*;
public interface MyCommand extends TargetableCommand {
    // Declare application methods here
}
```

---

b. Provide an implementation class for the interface.

Write a class that extends the `CacheableCommandImpl` class and implements your command interface. This class contains the code for the methods in your interface, the methods inherited from extended interfaces like the `CacheableCommand` interface, and the required or abstract methods in the `CacheableCommandImpl` class.

You can also override the default implementations of other methods provided in the `CacheableCommandImpl` class.

2. `cachespec.xml` must be configured.

The Trade3 home page uses the following commands. The command results (raw data) can be cached:

- Market summary command
- Account command
- Closed order command
- Holdings command

The `MarketSummaryCommand` is cached and invalidated on a timeout or by `ResetTradeCommand`, which uses the `MarketSummary` dependency ID as shown in Example 14-6.

#### Example 14-6 Cache entry for `MarketSummaryCommand`

---

```
<cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
  <name>com.ibm.websphere.samples.trade.command.MarketSummaryCommand
</name>
  <cache-id>
    <priority>3</priority>
    <timeout>10</timeout>
  </cache-id>
  <dependency-id>MarketSummary
</dependency-id>
</cache-entry>
```

---

The `AccountCommand` is used to cache a user's account information. `AccountCommands` are invalidated for each individual user when that users'



account information, such as their account balance, are updated by a trading operation. This is explained in Example 14-7:

*Example 14-7 Cache entry for AccountCommand*

---

```
<cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
  <name>com.ibm.websphere.samples.trade.command.AccountCommand</name>
  <cache-id>
    <component type="method" id="getUserID">
      <required>true</required>
    </component>
    <priority>3</priority>
  </cache-id>
  <dependency-id>Account_UserID
    <component id="getUserID" type="method">
      <required>true</required>
    </component>
  </dependency-id>
  <dependency-id>AllUsers
  </dependency-id>
</cache-entry>
```

---

The `OrderCompletedCommand` signifies that a buy or a sell order has completed for an individual user. When an order completes, a user's holdings and account balance have been modified. This invalidates the `AccountCommand`, `HoldingsCommand` and `OrdersCommand` for that user. See Example 14-8.

*Example 14-8 Cache entry for OrderCompletedCommand*

---

```
<cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
  <name>com.ibm.websphere.samples.trade.command.OrderCompletedCommand</name>
  <invalidation>Account_UserID
    <component id="getUserID" type="method">
      <required>true</required>
    </component>
  </invalidation>
  <invalidation>Holdings_UserID
    <component id="getUserID" type="method">
      <required>true</required>
    </component>
  </invalidation>
  <invalidation>Orders_UserID
    <component id="getUserID" type="method">
      <required>true</required>
    </component>
  </invalidation>
</cache-entry>
```

```

</invalidation>
<invalidation>ClosedOrders_UserID
  <component id="getUserID" type="method">
    <required>true</required>
  </component>
</invalidation>
</cache-entry>

```

---

The HoldingsCommand is used to cache a users' stock holdings information. HoldingsCommands are invalidated for each individual user when that user's holdings change due to an OrderCompletedCommand. It uses the Holdings\_UserID dependency ID to identify user Accounts by the userID. It also uses the AllUsers dependency ID to identify all user Accounts for commands which invalidate all Account Holdings cache entries. Refer to Example 14-9.

#### *Example 14-9 Cache entry for HoldingsCommand*

---

```

<cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
  <name>com.ibm.websphere.samples.trade.command.HoldingsCommand</name>
  <cache-id>
    <component type="method" id="getUserID">
      <required>true</required>
    </component>
    <priority>3</priority>
  </cache-id>
  <dependency-id>Holdings_UserID
  <component id="getUserID" type="method">
    <required>true</required>
  </component>
  </dependency-id>
  <dependency-id>AllUsers
  </dependency-id>
</cache-entry>

```

---

## Testing command result caching

We test the four commands included in the home page fragments, plus the home servlet and marketSummary.jsp:

1. Log into the Trade3 application:  
<http://appl.itso.ibm.com:9081/trade>
2. Open the Cache monitor:  
<http://appl.itso.ibm.com:9082/cachemonitor>
3. Clear the cache by clicking the **Clear Cache** button.

- Click the Home link in the Trade3 application and use the Cache monitor to verify that four commands, one servlet and one JSP are cached. If the caching works, you can see that six cache entries are used as shown in Figure 14-20 and Figure 14-21.

Statistic	Value
Cache Size	100
Used Entries	6
Cache Hits	0
Cache Misses	6
LRU Evictions	0
Explicit Removals	0
Default Priority	1
Servlet Caching Enabled	Yes
Disk Offload Enabled	No

Figure 14-20 Command result caching - Used Entries

Template	Cache ID
<a href="#">/trade/marketSummary.jsp</a>	<a href="#">/trade/app.action=home;JSESSIONID=0001okBFxqh2nfH1qe1xylJ-qBt:v3ayd613.ISO-8859-1</a>
<a href="#">/trade/marketSummary.jsp</a>	<a href="#">/trade/marketSummary.jsp.ISO-8859-1</a>
<a href="#">com.ibm.websphere.samples.trade.command.AccountCommand</a>	<a href="#">com.ibm.websphere.samples.trade.command.AccountCommand.getUserId=uid:0</a>
<a href="#">com.ibm.websphere.samples.trade.command.ClosedOrdersCommand</a>	<a href="#">com.ibm.websphere.samples.trade.command.ClosedOrdersCommand.getUserId=uid:0</a>
<a href="#">com.ibm.websphere.samples.trade.command.HoldingsCommand</a>	<a href="#">com.ibm.websphere.samples.trade.command.HoldingsCommand.getUserId=uid:0</a>
<a href="#">com.ibm.websphere.samples.trade.command.MarketSummaryCommand</a>	<a href="#">com.ibm.websphere.samples.trade.command.MarketSummaryCommand</a>

Figure 14-21 Command result caching - Cached entries

### 14.3.3 Cache replication

The cache replication mechanism allows data to be generated once and then be copied or replicated to other servers in the cluster, thus saving execution time and resources. Caching in a cluster has additional concerns. In particular, the same data could be required and hence be generated in multiple places. Also, the access to resources needed to generate the cached data can be restricted, preventing access to the data.

Cache replication also aids in cache consistency. If the cache entry is invalidated from one server, the invalidation is propagated to all servers.

The replication support uses a built-in high performance standard-based JMS messaging system as its underlying engine for data replication.

We continue to work with the redbook sample infrastructure, now configured for cache replication (Figure 14-22).

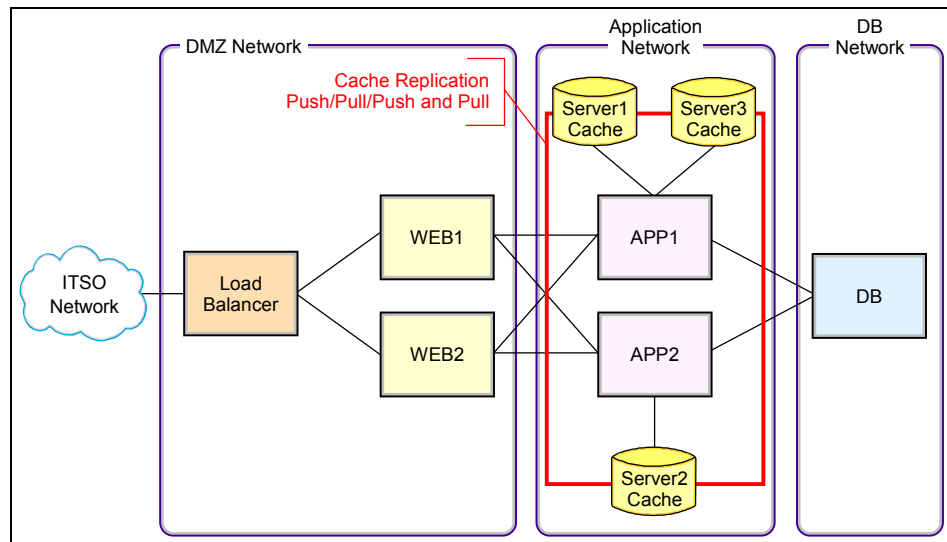


Figure 14-22 Redbook caching replication infrastructure

## Enable cache replication

The configuration specific to replication of data can exist as part of the Web container dynamic cache configuration accessible through the Administrative Console, or on a per cache entry basis through the `cachespec.xml` file. This includes the option to configure cache replication at the Web container level, but disabling it for a specific cache entry.

These configuration steps assume that there is already an internal replication domain and replicator entries defined on your system. If this is not true for your environment, then you first need to create these objects. Refer to the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195 for instructions on how to do this.

To enable cache replication you need to follow these steps:

1. Click **Servers -> Application Servers** in the Administrative Console navigation tree.

2. Select your application server.
3. Select **Dynamic Cache Service** from the Additional Properties.
4. Check the **Enable cache replication** check box (see Figure 14-13 on page 542) and click **Apply**.

## Configure cache replication

After you have enabled cache replication, you need to further configure it. Select the **Enable cache replication** link on the Dynamic Caching Services configuration panel to get into the cache replication configuration. Figure 14-23 is an example for Trade3Server1.

The advanced replication settings include fields for choosing the initial replicator entry that connects to the replicator domains. As an alternative, you can specify the IP addresses and ports (in the form address:port) for connections to replicators outside of the cell that the server is administered under. By default, if a replicator is defined on the server you are configuring, that server is the one chosen for cache replication. The replicator acts as a broker and distributes messages to the cluster.

**Important:** When adding a new server into an already existing cluster, you must also add the new server into the internal replication domain.

The screenshot shows a 'Configuration' dialog box with a 'General Properties' tab. It contains three main sections for configuration:

- Internal messaging server:**
  - Radio button selected: **Select replicator from the following domain**.
    - Domain: Trade3Cluster (dropdown)
    - Replicators: Trade3Server1 (dropdown)
  - Radio button unselected: **Select replicator from another domain**.
    - IP Address: (text field)
    - Port: (text field)
- Runtime mode:**
  - Push only (dropdown)
- Push frequency:**
  - 0 (text field) seconds

Help icons (i) are present next to the domain selection, runtime mode, and push frequency fields, providing additional context. At the bottom are buttons for Apply, OK, Reset, and Cancel.

Figure 14-23 Cache replication configuration

These are the Enable cache replication configuration options:

- ▶ Internal messaging server

Specifies a domain from which your data is replicated. You can choose any of the replicators defined under that domain. You can use the default domain or choose one from the drop-down menu. In our caching scenarios we configure replicators to be local to the server, for example a replicator for Trade3Server1 is Trade3Server1.

- ▶ Runtime mode

Specifies the global sharing policy for this server. The following settings are available:

- Push and Pull

In Push and Pull mode, cache entries are shared between application servers on demand. When an application server generates a cache entry, it broadcasts the cache ID of the created entry to all cooperating application servers. Each server then knows whether an entry exists for any given cache ID and for a given request for that entry, the application server knows whether to generate the entry or pull it from somewhere else. These entries cannot store non-serializable data.

In the scenario shown in Figure 14-24 on page 559, Trade3Server1 serves the cacheable/shareable object. It stores it in the Trade3Server1 cache. In addition, it also acts as a broker and sends a *cache ID* to the other servers in the cluster (steps *A<sub>n</sub>*). When Trade3Server3 receives a request for this published cache ID, it uses the cache ID to retrieve the object from the Trade3Server1 cache (steps *B<sub>n</sub>*).

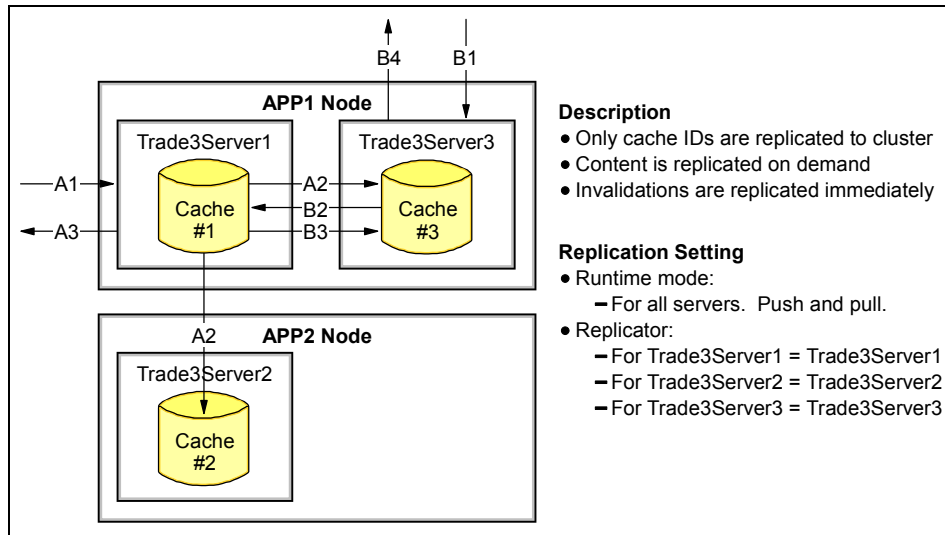


Figure 14-24 Pull and Push cache replication

An advantage of the Push and Pull option is that the content is distributed on demand. This has a positive effect on the total number of objects in the cache.

A small disadvantage is that retrieving the content from the remote cache can take longer comparing to the local cache. However, under normal circumstances it is significantly faster than re-generating the content.

#### – Push only

In Push only mode, the cache ID and cache content are automatically distributed to the dynamic caches in other application servers or cooperating JVMs. Each cache has a copy of the entry at the time it is created. These entries cannot store non-serializable data.

In the scenario shown in Figure 14-25 on page 560, Trade3Server1 serves the cacheable/shareable object. It stores it in the Trade3Server1 cache and it also acts as a broker and sends the *cache ID and the content* to other servers in the cluster (steps A<sub>n</sub>). When Trade3Server3 receives a request for this cache ID, it retrieves the object from its own cache (steps B<sub>n</sub>).

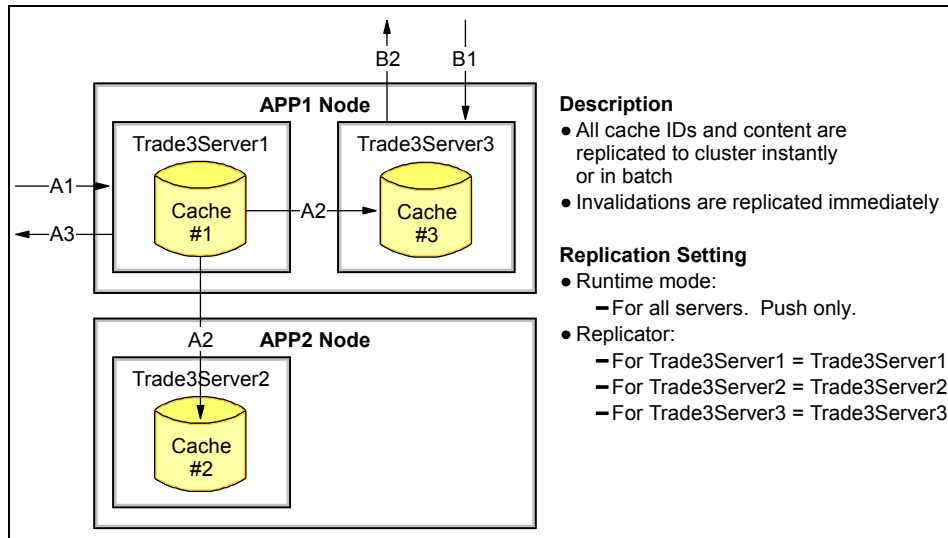


Figure 14-25 Push only cache replication

An advantage of the Push option is that all caches are synchronized, so even when Trade3Server1 crashes, Trade3Server3 does not need to regenerate the object again.

However, in the case when the total size of cached object is very high, synchronizing all objects can have a performance impact.

#### – Pull only

In Pull only mode, cache entries are shared between application servers on demand. If an application server gets a cache miss for an object, it queries the cooperating application servers to see if they have the object. If no application server has a cached copy of the object, the original application server executes the request and generates the object. These entries cannot store non-serializable data.

In the scenario shown in Figure 14-26 on page 561, Trade3Server1 gets a request for an object which is not in the Trade3Server1 local cache. It queries both Trade3Server2 and Trade3Server3 for this object, but none of them has the object in the cache. Therefore, Trade3Server1 executes the request and generates the object, which is cached in Trade3Server1's local cache (steps A<sub>n</sub>). Then Trade3Server3 receives a request for the same object and queries the cooperating application servers. This time the object is found in Trade3Server1's cache (steps B<sub>n</sub>).



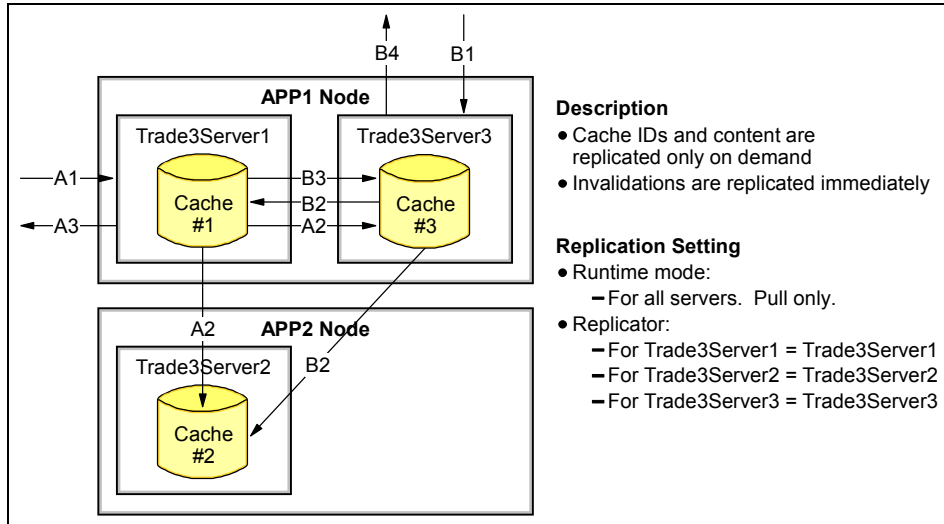


Figure 14-26 Pull only cache replication

#### – Not Shared

In Not Shared mode, cache entries are not shared among different application servers.

The default setting for a non-clustered environment is Not Shared. When enabling replication, the default value is Push only.

**Note:** To summarize the sharing policy, it is important to understand that the "push only" mode is good for workloads with a high probability of other clusters handling cache hits, "push and pull" is good for unknown or variable probability of cross-cluster cache hits, and "pull only" mode is good for low probability cache hits across the cluster.

#### ► Push frequency

Specifies the time in seconds to wait before pushing new or modified cache entries to other servers. A value of 0 (zero) means send immediately.

This is used by the dynamic cache service to provide a batch update option. Specifically, for the Push or Push and Pull options, the dynamic cache broadcasts the update asynchronously, based on a timed interval rather than sending them immediately. However, invalidators are sent immediately. Distribution of invalidations addresses the issue of stale data residing in a cluster.

Setting this property to a value greater than 0 (zero) causes a "batch" push of all cache entries that are created or modified during the time period.

## Configure cachespec.xml file for cache replication

The global replication configuration specified in the previous section applies for all caching objects unless the cachespec.xml configuration specifies otherwise. Every entry in cachespec.xml can use the *<sharing-policy>* element, with the values specified below. If the *<sharing-policy>* element is not present, the global settings are used.

The sharing policy on the cache entry level can have the same values as can be defined on the global level:

- ▶ not-shared
- ▶ shared-push
- ▶ shared-pull
- ▶ shared push-pull

Example 14-10 shows the *marketSummary.jsp* entry in *cachespec.xml*. This JSP will be cached and replicated based on the global settings because the *<sharing-policy>* element has not been specified.

*Example 14-10 Replication setting for marketSummary.jsp*

---

```
cache-entry>
  <class>servlet</class>
  <name>/marketSummary.jsp</name>
  <property name="EdgeCacheable">true</property>
  <cache-id>
    <priority>3</priority>
    <timeout>180</timeout>
  </cache-id>
  <dependency-id>MarketSummary
  </dependency-id>
</cache-entry>
```

---

Example 14-11 on page 563 contains an entry for *MarketSummaryComand*. This command result will not be replicated, because the *sharing-policy* element value explicitly specifies that this object cannot be shared among different application servers.

**Note:** Even though this entry is not shared, in a replicated environment, invalidations are still sent.

```
<cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
  <name>com.ibm.websphere.samples.trade.command.MarketSummaryCommand</name>
  <cache-id>
    <priority>3</priority>
    <timeout>10</timeout>
  </cache-id>
  <dependency-id>MarketSummary
  </dependency-id>
</cache-entry>
```

---

## Cache replication testing

We tested all 3 replication options described in this chapter.

However, we had to limit the “pull” testing to replicate between two servers only because the pull functionality among more than two servers was not available at the time we wrote this book.

### 14.3.4 Cache invalidation

The difference between caching static and dynamic content is the requirement for proactive and effective invalidation mechanisms to ensure the freshness of content. The time-based invalidation alone is no longer adequate for dynamic cache invalidation.

The dynamic cache service provides event-based and time-based invalidation techniques. WebSphere Application Server V5.1 offers access to programmatic cache and invalidation techniques. Invalidation policies can be defined with XML cache policy files. Invalidation policies allow triggering events to invalidate cache entries without the need to write explicit code. More complex invalidation scenarios may require code, which invokes the invalidation API.

Example 14-12 shows the invalidation policy, which invalidates cache entries for groups of objects using the same `Account_UserID` dependency ID. For example, home servlet and all commands invoked by home servlet are invalidated when the user logs out from Trade3 application.

```
<cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
  <name>com.ibm.websphere.samples.trade.command.LogoutCommand</name>
  <invalidation>Account_UserID
```

```

        <component id="getUserID" type="method">
            <required>true</required>
        </component>
    </invalidation>
    <invalidation>Holdings_UserID1
        <component id="getUserID" type="method">
            <required>true</required>
        </component>
    </invalidation>
</cache-entry>

```

---

We also show an example for time based invalidation in Example 14-4 on page 549, where marketSummary.jsp has a timeout value of 180 seconds.

**Note:** In order to use caching safely, you need to make sure that you configure invalidation properly. To do this, you must be familiar with the specifications of the cachespec.xml file and the way how invalidation is handled in your application's code. This chapter provides you with one example, but please refer to the WebSphere InfoCenter for more detailed information on the cachespec.xml file.

If you use cache replication, please note that invalidations are always sent immediately regardless of the share type defined in cachespec.xml.

### 14.3.5 Troubleshooting the dynamic cache service

In case you are experiencing problems related to the WebSphere Dynamic Cache service, you should follow these steps to resolve your problem:

1. Use the Cache monitor to test.
2. Verify cachespec.xml file.
3. Review the JVM logs for your application server. Messages prefaced with DYNA result from dynamic cache service operations.
  - a. View the JVM logs for your application server. Each server has its own JVM log file. For example, if your server is named Member\_1, the JVM log is located in the subdirectory <install\_root>/logs/Member\_1/.

Alternatively, you can use the Administrative Console to review the JVM logs, click **Troubleshooting -> Logs and Trace -> <server\_name> -> JVM Logs -> Runtime tab -> View.**

- b. Find any messages prefaced with DYNA in the JVM logs, and write down the message IDs. A sample message having the message ID DYNA0030E follows:
 

DYNA0030E: "property" element is missing required attribute "name".
  - c. Find the message for each message ID in the WebSphere Application Server InfoCenter. In the InfoCenter navigation tree, click **<product\_name> -> Reference -> Messages -> DYNA** to view dynamic cache service messages.
  - d. Try the solutions stated under User Action in the DYNA messages.
4. Enable tracing on the application server for `com.ibm.ws.cache.*`. To enable it, click **Troubleshooting -> Logs and Trace -> <server\_name> -> Diagnostic Trace -> Configuration tab**. You can then configure the tracing options as shown in Figure 14-27.

If you experience problems with cache replication, then you need to enable the trace for `com.ibm.ws.drs.*`.

You need to restart the server and monitor the trace file.

Configuration		Runtime
<b>General Properties</b>		
Enable Trace	<input checked="" type="checkbox"/> Enable trace with the following specification	<i>i</i> Check this box to enable the selected trace service.
Trace Specification	<input type="text" value="com.ibm.ws.cache.*all=enabled"/> <input type="button" value="Modify..."/>	<i>i</i> Use these options to specify tracing details.
Trace Output	<input type="radio"/> Memory Buffer Maximum Buffer Size * <input type="text" value="8"/> thousand entries  <input checked="" type="radio"/> File Maximum File Size * <input type="text" value="20"/> MB Maximum Number of Historical Files * <input type="text" value="1"/> File Name * <input type="text" value="\${SERVER_LOG_ROOT}/trace_dynacache.log"/>	<i>i</i> Use these options to specify the type of output generated by the trace.
Trace Output Format	<input type="text" value="Advanced"/>	<i>i</i> Use this field to specify the format of the trace output.
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>		

Figure 14-27 Dynamic cache tracing configuration

## 14.4 WebSphere external caching scenarios

The objectives of our external caching scenarios are:

- ▶ Describe the external caching options
- ▶ Configure and test external caching options
- ▶ Explore cache invalidation

We use the following external caches, which can be controlled by the WebSphere dynamic cache service:

- ▶ Web server plug-in

In addition to the well known Web server plug-in functionalities (such as failover and load balancing), the Web server plug-in is integrated with the WebSphere Dynamic Cache to provide in-memory caching of servlets and JSP pages. It uses the ESI fragment assembly, which further enhances this caching ability with on-the-fly reassembly of JSP pages.

- ▶ IBM HTTP Server's high-speed cache

This cache is referred to as the *Fast Response Cache Accelerator* (FRCA) to cache whole pages and fragments.

- ▶ Edge Components of WebSphere Application Server Network Deployment V5.x

The Edge Components can also be configured as WebSphere Application Server's external cache for whole page caching. In this case, the dynamic cache service can be enabled to match pages with their universal resource identifiers (URIs) and export matching pages to the external cache (Figure 14-28 on page 567). The contents can then be served from the external cache instead of the application server to significantly save resources and improve performance.

Figure 14-28 on page 567 and Figure 14-29 on page 567 show two examples of exporting a dynamic cache page from the WebSphere Application Server to the external cache.

The first example (Figure 14-28 on page 567) shows caching of whole pages, where page A.jsp consists of three fragments. These three JSPs are included at compilation time. The event-driven invalidation causes the invalidation of A.jsp from the external cache as a result of any update to B.jsp, C.jsp, or D.jsp.

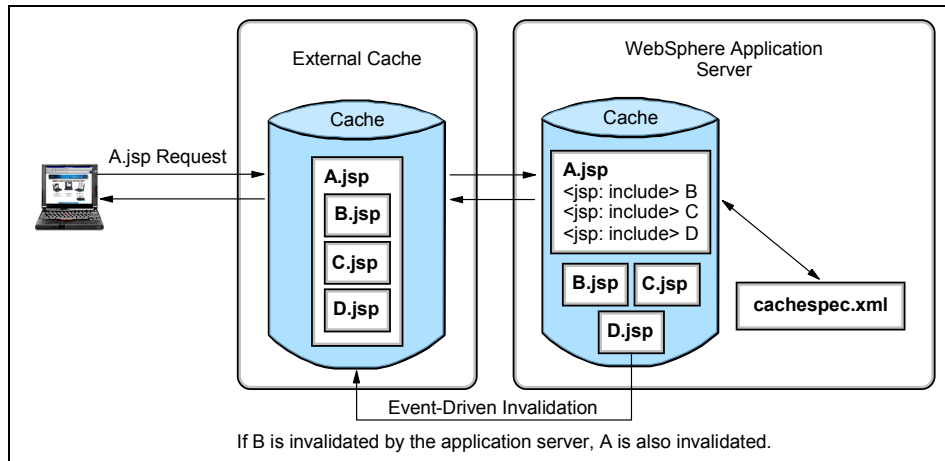


Figure 14-28 Whole page external caching

The second example (Figure 14-29) shows caching of a whole page and its' fragments using Edge Side Include (ESI). In this case, if a fragment (for example B.jsp) is invalidated by event-driven or time-driven invalidation, then the page A.jsp is not invalidated and ESI will tell the cache server to fetch only the invalidated fragment (B.jsp) from the application server. Then, the external cache using the ESI processor assembles a new page A.jsp which consists of the old fragments C.jsp and D.jsp and the new fragment B.jsp.

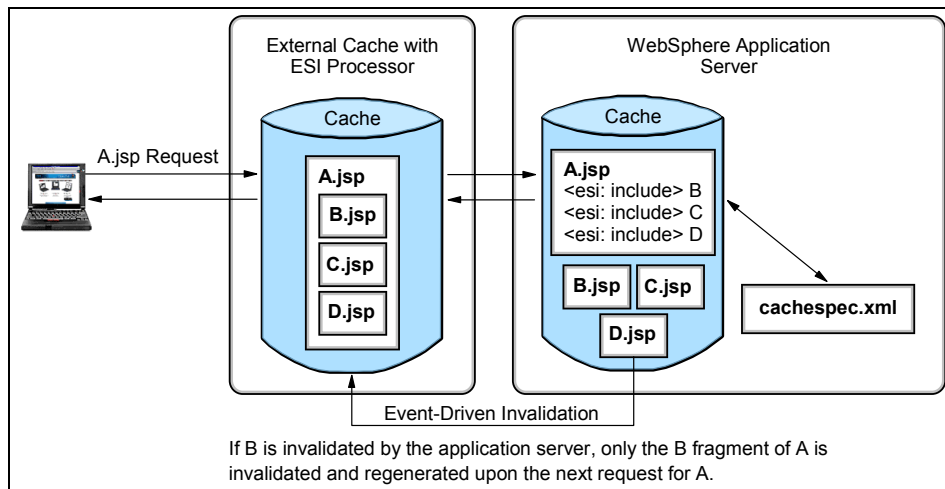


Figure 14-29 Fragment external caching

For example, the Trade3 home servlet contains fragment `marketSummary.jsp`. If we invalidate `marketSummary.jsp`, only this JSP will be regenerated upon the next request for the home servlet.

Any servlet and JSP file content that is private, requires authentication, or uses SSL should not be cached externally. The authentication required for those servlet or JSP file fragments cannot be performed on the external cache. A suitable timeout value should be specified if the content is likely to become stale.

### 14.4.1 WebSphere External Cache configuration

While working on our scenarios we need to perform a few configurations. We explain these configuration steps in this section at the generic level and refer back to this section from each scenario.

#### External cache group settings

You need to define an external cache group controlled by WebSphere Application Server. To do so:

1. Start the WebSphere Administrative Console and click **Servers -> Application Servers**.
2. Select your application server from the list.
3. Select **Dynamic Cache Service** from the Additional Properties.
4. Click **External Cache Groups**, again from the Additional Properties pane.

This panel (shown in Figure 14-30 on page 569) allows you to create, delete or update an existing External Cache Group.

5. Click **New**. This launches the Configuration window where you can specify the name of the external cache group.

**Note:** The external cache group name needs to match the `externalcache` property defined for servlets or JSPs in the `cachespec.xml` file.

When external caching is enabled, the cache matches pages with its URIs and pushes matching pages to the external cache. The entries can then be served from the external cache instead of the application server.

We need to create three different external cache groups for our scenarios:

- ▶ AFPA - which is the adapter used for the Fast Response Cache Accelerator
- ▶ EsInvalidator - used for the Web server plug-in
- ▶ ITSO-APPSERV - used for the Caching Proxy



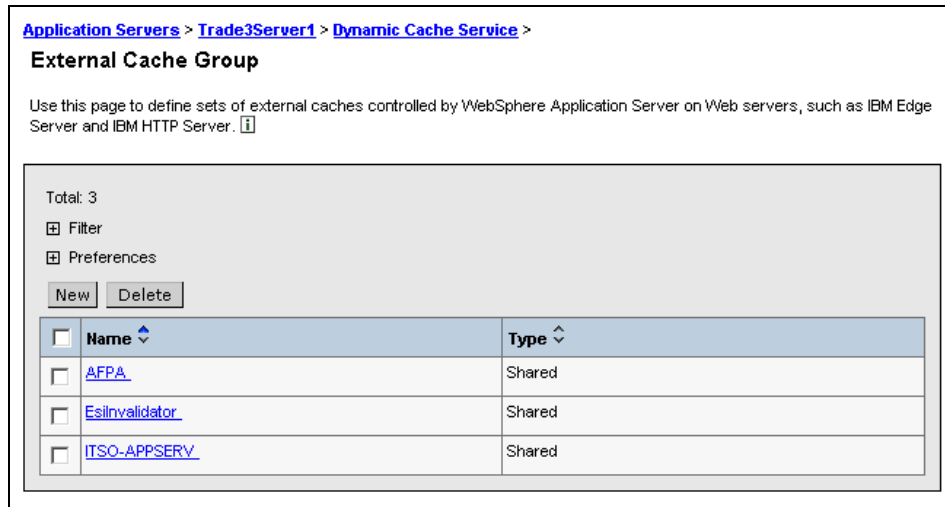


Figure 14-30 External Cache Groups

## External cache group member settings

Once you have created the three external groups, you need to configure the specific caches that are members of each cache group.

1. Open the Administrative Console and click **Servers -> Application Servers**.
2. Select your application server.
3. Select **Dynamic Cache Service** from the Additional Properties, then **External Cache Groups**.
4. Click **<external\_cache\_group\_name>** followed by **External Cache Group Members** from the Additional Properties.
5. Click **New** to launch the Configuration panel.

There are two values that you need to specify on the Configuration panel:

- Address

Specifies a configuration string used by the external cache adapter bean to connect to the external cache.

- Adapter Bean Name

Specifies an adapter bean specific for the external cache member.

You can configure three different types of external cache group members.

- Fast Response Cache Accelerator (called AFPA in our example):
  - Address: Hostname:port (hostname and port on which AFPA listens)

- Adapter Bean Name: `com.ibm.ws.cache.servlet.Afpa`
- ▶ ESI (called `EsIInvalidator` in our scenario):
  - Address: Hostname (and possibly port number)
  - Adapter Bean Name:  
`com.ibm.websphere.servlet.cache.ESIInvalidatorServlet`
- ▶ WTE - Caching Proxy (called `ITSO-APPSERV` in our example):
  - Address: `hostname:port` (hostname and port on which WTE is listening)
  - Adapter Bean Name: `com.ibm.websphere.edge.dynacache.WteAdapter`

### 14.4.2 External caching by Web server plug-in

WebSphere Application Server leverages the Edge Side Includes (ESI) specification to enable caching and assembly of distributed fragments.

Edge Side Includes is a simple mark-up language used to define Web page fragments for dynamic assembly of a Web page at the edge of network. ESI is an open standard specification supported by Akamai and leaders in the Application Server, Content Management Systems, and Content Delivery Networks markets.

With the Distributed Fragment Caching and Assembly Support, WebSphere Application Server customers can defer page assembly to any ESI compliant surrogate server, such as Akamai EdgeSuite service. This may result in a significant performance advantage if fragments can be cached and reused at the surrogate.

WebSphere Application Server provides distributed fragment caching and assembly support through the Web server plug-in. WebSphere Application Server uses the Web server plug-in to communicate with the HTTP Server. This plug-in has the ability to cache whole pages or fragments. Additionally, it can dynamically assemble Web pages containing ESI `<esi:include>` tags. Any server with WebSphere HTTP plug-in support can gain the benefits provided by the WebSphere Application Server dynamic cache service. Figure 14-31 on page 571 show the infrastructure topology we used for this scenario. As you can see here, the edge cache resides on the Web server machines:

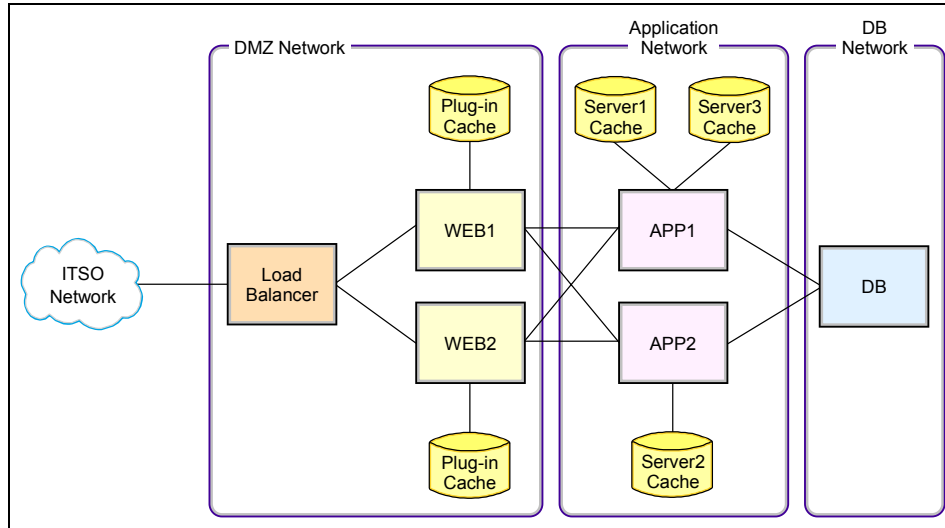


Figure 14-31 ITSO Web server plug-in infrastructure

With dynamic cache service's external cache control, distributed fragment caching and assembly support, dynamic content can be exported, cached, and assembled at the most optimal location, closer to the end user. More important, WebSphere Application Server can maintain control of the external cache through its Invalidation Support to ensure the freshness of cached content. As a result, WebSphere Application Server customers are equipped to create and serve highly dynamic Web pages without jeopardizing page performance and user experiences.

In “Edge Side Includes” on page 588 we provide some existing ESI benchmarks for the Trade3 sample application.

## ESI Processor

The cache implemented by the ESI processor is an in-memory cache, not a disk cache. Therefore, the cache entries are not saved when the Web server is restarted.

The basic operation of the ESI processor is as follows: When a request is received by the Web server plug-in, it is sent to the ESI processor, unless the ESI processor is disabled. It is enabled by default. If a cache miss occurs, a Surrogate-Capabilities header is added to the request and the request is forwarded to the WebSphere Application Server. If the dynamic servlet cache is enabled in the application server, and the response is edge cacheable, the application server returns a Surrogate-Control header in response to the WebSphere Application Server plug-in.

The value of the `Surrogate-Control` response header contains the list of rules that are used by the ESI processor in order to generate the cache ID. The response is then stored in the ESI cache, using the cache ID as the key. For each ESI include tag in the body of the response, a new request is processed such that each nested include results in either a cache hit or another request forwarded to the application server. When all nested includes have been processed, the page is assembled and returned to the client.

## Configuration steps

1. Configure the external cache group on the application servers (in our case Trade3Server1, Trade3Server2 and Trade3Server3). Follow the steps outlined in “WebSphere External Cache configuration” on page 568 and create an external cache group named “EsiInvalidator”. Then add a member with these parameters:

- Address: Local host
- Adapter bean name:  
`com.ibm.websphere.servlet.cache.ESIInvalidatorServlet`

2. Configure the ESI processor.

The ESI processor is configurable through the WebSphere Web server plug-in configuration file (`plugin-cfg.xml`). Example 14-13 shows the beginning of this file, which illustrates the ESI configuration options.

*Example 14-13 ESI configuration options in plugin-cfg.xml file*

---

```
<Config>
<Property Name="esiEnable" Value="true"/>
<Property Name="esiMaxCacheSize" Value="1024"/>
<Property Name="esiInvalidationMonitor" Value="false"/>
```

---

The `esiMaxCacheSize` specifies the maximum size of the cache in 1 KB units. The default maximum size of the cache is 1 MB. If the cache is full, the first entry to be evicted from the cache is the entry that is closest to expiration.

The `esiInvalidationMonitor` parameter specifies whether or not the ESI processor should receive invalidations from the application server.

3. In order to enable an application server to send an explicit invalidation, the `esiInvalidationMonitor` property from Example 14-13 must be set to true and the `DynaCacheEsi` application must be installed on the application server. The `DynaCacheEsi` application is located in the `installableApps` directory and is named `DynaCacheEsi.ear`.

If the `esiInvalidationMonitor` property is set to true but the `DynaCacheEsi` application is not installed, then errors will occur in the Web server plug-in and the request will fail.

4. Add a cache policy in the cachespec.xml file for the servlet or JSP file you want to be cached in the edge cache. To do so, Add <property name="EdgeCacheable">true</property> for those entries, as explained in Example 14-14 for the home servlet.

*Example 14-14 Cache ID for home servlet*

---

```
<cache-id>
  <component id="action" type="parameter">
    <value>home</value>
    <required>true</required>
  </component>
  <component id="JSESSIONID" type="cookie">
    <required>true</required>
  </component>
  <property name="EdgeCacheable">true</property>
</cache-id>
```

---

5. When WebSphere Application Server is used to serve static data, such as images and HTML on the application server, the URLs are also cached in the ESI processor. This data has a default timeout of 300 seconds. You can change the timeout value by adding the property com.ibm.servlet.file.esi.timeOut to your JVM's command-line parameters.

## ESI cache monitoring

The ESI processor's cache is monitored through the Dynamic Cache Monitor application. In order for ESI processor's cache to be visible in the Cache monitor, the DynaCacheEsi application must be installed and the esiInvalidationMonitor property must be set to true in the plugin-cfg.xml file.

To get this statistic you need to select **Edge Statistics** in the Cache monitor navigation bar.

The Edge Statistics could be confused with the Caching Proxy statistics, which is part of IBM Edge Components. However, the term Edge statistics in this case relates to the ESI cache statistics. The ESI cache statistics are shown in Figure 14-32 on page 574.

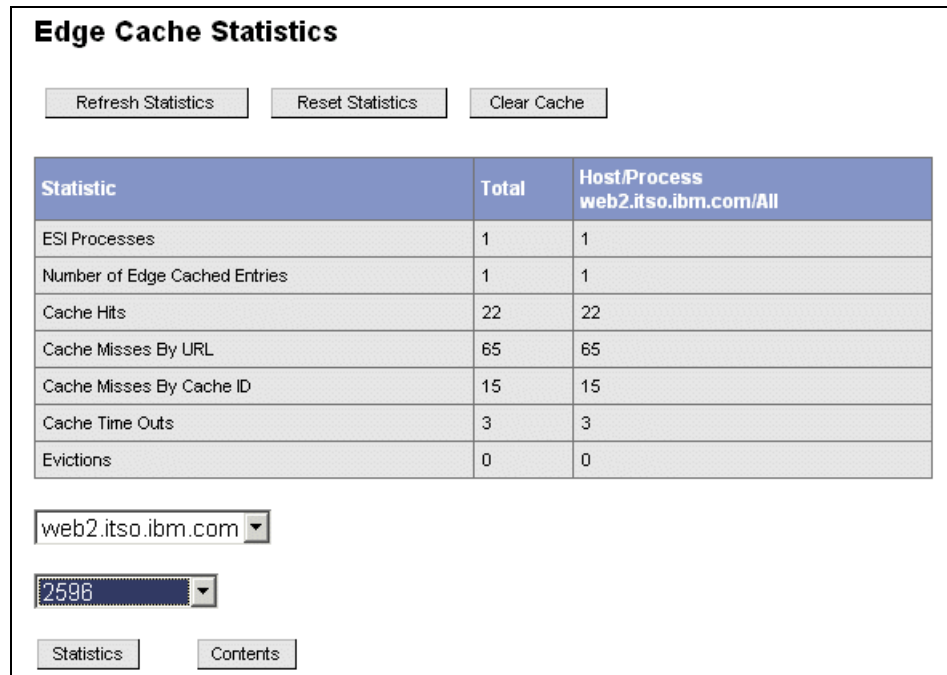


Figure 14-32 ESI statistics (Edge statistics)

The following information is available:

- ▶ **ESI Processes:** This is the number of processes configured as edge caches. In our example in Figure 14-32, you can see that we have one process running on the was2.itso.ibm.com server. The process id is 2708 (see Figure 14-33 on page 575).

- ▶ **Number of Edge Cached Entries:** This is the number of entries currently cached on all edge servers and processes.

- ▶ **Cache Hits:** This is the number of requests that match entries on edge servers.

- ▶ **Cache Misses By URL:** A cache policy does not exist on the edge server for the requested template.

Note that the initial ESI request for a template that has a cache policy on a WebSphere Application Server results in a miss. Every request for a template that does not have a cache policy on the WebSphere Application Server will result in a miss by URL on the Edge server.

- ▶ **Cache Misses By Cache ID:** In this case, the cache policy for the requested template exists on the edge server. The cache IDs is created (based on the ID rules and the request attributes), but the cache entry for this ID does not exist.

Note, that if the policy exists on the edge server for the requested template, but a cache ID match is not found, the request is not treated as a cache miss.

- **Cache Time Outs:** The number of entries removed from the edge cache, based on the timeout value.
- **Evictions:** The number of entries removed from the edge cache, due to invalidations received from WebSphere Application Server.

If you click the **Content** button, you can see the edge content for all processes or for a selected process number. You can see an example in Figure 14-33.

**Current Edge Cache Contents for Host: web2.itso.ibm.com, Process: 2,708**

Entries 1 through 11 of 11  
<0>

Refresh Contents

Clear Cache

Cache ID	Host	Process
GET_trade/app_action=account:JSESSIONID=0001Dk9gob1fKyTnKiUc_yAB4FI:v3avd52a	web2.itso.ibm.com	2708
GET_trade/app_action=home:JSESSIONID=0001Dk9gob1fKyTnKiUc_yAB4FI:v3avd52a	web2.itso.ibm.com	2708
GET_trade/app_action=portfolio:JSESSIONID=0001Dk9gob1fKyTnKiUc_yAB4FI:v3avd52a	web2.itso.ibm.com	2708
GET_trade/app_action=quotes:symbols=s:0,s:1,s:2,s:3,s:4	web2.itso.ibm.com	2708
GET_trade/displayQuote.jsp_symbol=s:0	web2.itso.ibm.com	2708
GET_trade/displayQuote.jsp_symbol=s:1	web2.itso.ibm.com	2708
GET_trade/displayQuote.jsp_symbol=s:2	web2.itso.ibm.com	2708
GET_trade/displayQuote.jsp_symbol=s:3	web2.itso.ibm.com	2708
GET_trade/displayQuote.jsp_symbol=s:4	web2.itso.ibm.com	2708
GET_trade/marketSummary.jsp	web2.itso.ibm.com	2708
POST_trade/marketSummary.jsp	web2.itso.ibm.com	2708

Figure 14-33 Edge cache content

## Cache invalidation

There are three methods by which entries are removed from the ESI cache:

1. An entry's expiration timeout could fire.
2. An entry may be purged to make room for newer entries.
3. The application server could send an explicit invalidation for a group of entries.

### 14.4.3 External caching on IBM HTTP Server

#### Restriction:

- ▶ The Fast Response Cache Accelerator (FRCA) is available for both Windows NT and Windows 2000 operating systems and AIX platforms. However, the dynamic cache is only available on the Windows operating systems.
- ▶ At the time of writing this redbook, afpaplugin.dll was not available for IHS2.0. This dll extends FRCA and enables the WebSphere Application Server to directly interface/access the FRCA, which is needed for dynamic caching on IHS.

WebSphere Application Server fragment cache can use the IBM HTTP Server as an external cache. We explain the appropriate configuration in this section. However, we were not able to test this scenario because our redbook sample configuration uses IBM HTTP Server 2.0 and afpaplugin.dll was not yet available for this version while writing this redbook.

The FRCA cache resides on the Web server node as shown in Figure 14-34.

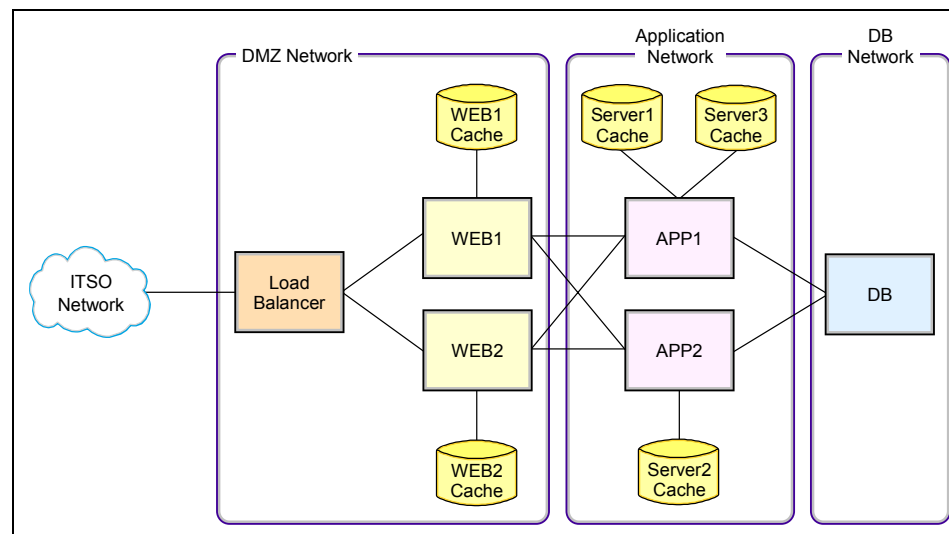


Figure 14-34 ITSO IBM HTTP Server infrastructure

#### Configuration steps

1. Configure the external cache group on the application servers (in our case Trade3Server1, Trade3Server2, and Trade3Server3). Follow the steps



outlined in “WebSphere External Cache configuration” on page 568 and create an external cache group called “AFPA”. Then add a member with these parameters:

- Address: hostame:port
  - Adapter bean name: com.ibm.ws.cache.servlet.Afpa
2. Add a cache policy to the cachespec.xml file for the servlet or JSP file you want to cache. In our example we added
- ```
<property name="ExternalCache">afpa</property>
```
- for all entries that should be cached by FRCA.

---

*Example 14-15 Cache entry for marketSummary.jsp*

---

```
<cache-entry>
  <class>servlet</class>
  <name>/marketSummary.jsp</name>
  <property name="ExternalCache">afpa</property>
  <cache-id>
    <priority>3</priority>
    <timeout>180</timeout>
  </cache-id>
  <dependency-id>MarketSummary
  </dependency-id>
</cache-entry>
```

---

3. Modify the IBM HTTP Server configuration and add the following directives at the end of the httpd.conf file (see Example 14-16):

---

*Example 14-16 AFPA adapter configuration in httpd.conf file*

---

```
AfpaEnable
AfpaCache on
AfpaLogFile "<install_root>\IBMHttpServer\logs\afpalog" V-ECLF
LoadModule afpaplugin_module <install_root>/bin/afpaplugin.dll
AfpaPluginHost WAS_Hostname:port
```

---

AfpaPluginHost WAS\_Hostname:port specifies the host name and port of the application server. These are the same values as you have specified in the Address field when configuring the external cache group member (step 1 on page 576).

The LoadModule directive loads the IBM HTTP Server plug-in that connects the Fast Response Cache Accelerator to the WebSphere Application Server fragment cache.

- If multiple IBM HTTP Servers are routing requests to a single application server, add the directives from Example 14-16 to the httpd.conf file of each of these IBM HTTP Servers for distributed platforms.

- If one IBM HTTP Server is routing requests to a cluster of application servers, add the `AfpaPluginHost WAS_Hostname:port` directive to the `httpd.conf` file for each application server in the cluster. For example, if there are three application servers in the cluster (`Trade3Server1`, `Trade3Server2`, `Trade3Server3`) add an `AfpaPluginHost` directive for each application server as shown in Example 14-17):

*Example 14-17 AFPA adapter configuration in httpd.conf file for application server cluster*

---

```
AfpaPluginHost Trade3Server1:9001
AfpaPluginHost Trade3Server1:9002
AfpaPluginHost Trade3Server1:9003
```

---

### Cache monitoring

You can monitor the FRCA cache log file. The location of this log file is also configured in the `httpd.conf` file (`AfpaLogFile` directive - see Example 14-16 on page 577).

## 14.4.4 External caching on Caching Proxy

The dynamic caching function enables the Caching Proxy to cache dynamically generated content in the form of responses from JSPs and servlets generated by IBM WebSphere Application Server. A Caching Proxy adapter module is used at the application server to modify the responses, so that they can be cached at the proxy server in addition to being cached in the application server's dynamic cache. With this feature, dynamically generated content can be cached at the entry point to the network, avoiding repeated requests to the application server, when the same content is requested by multiple clients.

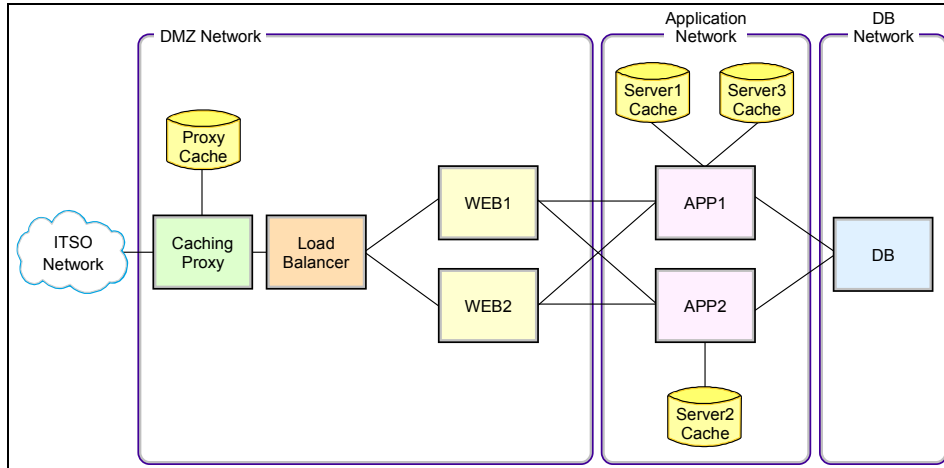


Figure 14-35 ITSO Caching Proxy infrastructure

The caches on the Caching Proxy and the WebSphere Application Server(s) are synchronized when a new result is generated or a cache object expires. Cached dynamic pages do not expire in the same way that regular files do. They can be invalidated by the application server that generated them.

## Configuration steps

1. Configure the external cache group on the application servers (in our case Trade3Server1, Trade3Server2 and Trade3Server3). Follow the steps outlined in “WebSphere External Cache configuration” on page 568 and create an external cache group. Then add a member with these parameters:
  - Address: hostname:port
  - Adapter bean name: `com.ibm.websphere.edge.dynacache.WteAdapter`

We have created a cache group called ITSO-APPSERV (Figure 14-30 on page 569) and added our Caching Proxy server to this group (see Figure 14-36 on page 580).

Configuration		
General Properties		
Address	* cproxy.itso.ibm.com	The hostname and possibly port number of an external cache.
Adapter Bean Name	* com.ibm.websphere.edge.dynacach	The name of a class, located on the WebSphere Application Server classpath, of the adapter between WebSphere Application Server and this external cache.
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>		

Figure 14-36 ITSO-APPSERV external cache group member setting

2. Modify the dynaedge-cfg.xml file (in <install\_root>/AppServer/properties) and add the external group statement as shown in Example 14-18.

Example 14-18 dynaedge-cfg.xml file

---

```

<EdgeServerCfg CacheManager="ITSO-APPSERV">
<EdgeServer
    endpoint = "http://cproxy.itso.ibm.com:80"
    user = "webadmin"
    userPasswd = "webadmin"
    invalidation-url = "/WES_External_Adapter"
    URI-type= "absolute"
/>

```

---

The log file <install\_root>/appserver/logs/edge/logs/dynacache.log can be used to verify that your dynamic cache adapter was initialized properly.

3. Add a cache policy to the cachespec.xml file for the servlet or JSP file you want to cache. Add <property name="ExternalCache">ITSO-APPSERV</property> for all entries which should be cached by the proxy cache server.

Example 14-19 Cache ID for home servlet

---

```

<cache-id>
  <component id="action" type="parameter">
    <value>home</value>
    <required>true</required>
  </component>
  <component id="JSESSIONID" type="cookie">
    <required>true</required>
  </component>
  <property name="ExternalCache">ITSO-APPSERV</property>
</cache-id>

```

---

#### 4. Configure dynamic caching at the Caching Proxy server.

The configuration changes are done in the caching proxy configuration file `ibmproxy.conf`. This file is stored in the `<install_root>/ibm/edge/cp/etc/en_US` directory.

- Set the `Service` directive to enable the dynamic caching plug-in as shown in Example 14-20. Note that each directive must appear on a single line in the proxy configuration file.

##### *Example 14-20 Service directive*

---

```
# ===== JSP Plug-in =====  
Service /WES_External_Adapter  
/opt/ibm/edge/cp/lib/plugins/dynacache/libdyna_plugin.o:exec_dynacmd
```

---

- Set the `ExternalCacheManager` directive to specify file sources. Each Caching Proxy must also be configured to recognize the source of the dynamically generated files. See Example 14-21. You need to add an `ExternalCacheManager` directive to the `ibmproxy.conf` file for each application server that caches dynamically generated content at this proxy server. This directive specifies a WebSphere Application Server or the cluster of application servers that caches results at the proxy, and sets a maximum expiration time for content from that server/cluster.

##### *Example 14-21 ExternalCacheManager directive*

---

```
#ExternalCacheManager directive:  
ExternalCacheManager ITS0-APPSERV 20 minutes
```

---

- Next you need to add a `CacheQueries` directive to enable cache responses for requests that contain a question mark (?), for example `/trade/app?action=home`. See Example 14-22.

##### *Example 14-22 CacheQueries directive*

---

```
#CacheQueries directive:  
CacheQueries ALWAYS http://cluster.itso.ibm.com/*
```

---

- Set the `CacheTimeMargin`, which specifies the minimum expiration time; files with expiration times below this minimum are not cached. Because query responses sometimes have very short expiration times, setting this directive to a lower setting allows more query responses to be cached. See Example 14-23 on page 582.

#### Example 14-23 CacheTimeMargin directive

---

```
# CacheTimeMargin directive:
CacheTimeMargin 1 minutes
```

---

For more information on configuring dynamic caching in the Caching Proxy, refer to the *Caching Proxy Administration Guide*, GC09-4601, available on the Internet at:

<http://www.ibm.com/software/webservers/appserv/doc/v50/ec/infocenter/index.html>

### Caching Proxy monitoring

There is no GUI interface available for monitoring the cache content on the Caching Proxy server. Therefore, you need to look at some log files to understand what is cached and what is not. The relevant log files are:

- ▶ ProxyAccessLog - used for logging proxy requests
- ▶ CacheAccessLog - used for logging hits on proxy cache

The file location can be specified in the `ibmproxy.conf` file. The default location is the `<install_root>/ibm/edge/cp/server_root/logs/` directory.

### Caching Proxy troubleshooting

You have two options to delete the Caching Proxy cache:

- ▶ If you are running with a memory cache only, then the only way to purge the cache is to restart the process.
- ▶ If you are running with a raw disk cache, then you can use the **htcformat** command to reformat the cache device. This command is available on AIX, Windows and Linux platforms.

You can enable tracing for the Caching Proxy. To do this, you need to specify the log file for tracing in the `ibmproxy.conf` file as shown in Example 14-24.

#### Example 14-24 Tracing log file

---

```
#log file directives
TraceLog /opt/ibm/edge/cp/server_root/logs/trace
```

---

Also, you can enable tracing on the application server for `com.ibm.websphere.edge.*`, which is the package containing the external cache adapters. To do this, click **Troubleshooting -> Logs and Trace -> <server\_name> -> Diagnostic Trace** in the WebSphere Administrative Console. Enter the trace configuration options on the **Configuration tab**. An example of this window is found in Figure 14-27 on page 565.

### 14.4.5 External cache invalidation

The responsibility for synchronizing the dynamic cache of external caches and the WebSphere Application Server is shared by both systems. For example, a public Web page dynamically created and cached at the application server using Servlet/JSP result caching can be exported by the application server and cached by the external cache server, which can serve the application's execution results repeatedly to many different users until notified that the page is invalid.

The content in the external cache is valid until:

- ▶ The proxy server removes an entry because the cache is congested.
- ▶ Or the default timeout set by the Caching Proxy's configuration file expires.
- ▶ Or the Caching Proxy receives an Invalidate message directing it to purge the content from its cache. Invalidate messages originate at the WebSphere Application Server that owns the content and are propagated to each configured Caching Proxy.

## 14.5 Conclusion

As more e-business sites seek to retain customers by serving personalized content, they face potential server-side bottlenecks, slow user response time, and increasing infrastructure costs. The WebSphere Application Server dynamic cache service can solve these critical challenges. Caching dynamic content that needs back-end requests or CPU-intensive computations can reduce server-side bottlenecks and maximize system resources, thus boosting performance and reducing infrastructure costs.

The WebSphere Application Server dynamic cache service is easy to use and readily available. You can benefit from using the available comprehensive functions for caching dynamic content. The Servlet/JSP result cache and command result cache make the caching of dynamic content possible - at various levels of granularity for the highest possible cache hits. The replication and invalidation support facilitates caches that can be shared, replicated, and synchronized in multi-tier or multiserver environments. The Edge of Network Caching Support, with its external caches and fragment support, generates a virtual extension of application server caches into the network.

The WebSphere Application Server dynamic cache service combined with an appropriate external cache, such as WebSphere Edge Server or IBM HTTP Server, can power high volume sites with intensive dynamic content to achieve the highest level of scalability and performance.

The last information in this chapter is a short recapitulation of the caching technologies available for WebSphere Application Server solutions. Refer to Table 14-1.

*Table 14-1 Caching techniques*

	Middleware	What is cached	Memory or file system
Dynamic Caching	WebSphere Application Server	Results from JSPs Servlets	Both
Fast Response Cache Accelerator <sup>a</sup>	IBM HTTP Server <sup>b</sup>	Results from JSP Servlets	V1.3 - Memory V2.0 - Both <sup>c</sup>
Web server plug-in	WebSphere Application Server - ESI processor	Results from JSP Servlets	Memory
Caching Proxy	WebSphere Edge Components	Results from JSP Servlets	Both

a. The Fast Response Cache Accelerator is available on Windows NT and Windows 2000 operating systems and the AIX platform. However, support to cache dynamic content is only available on the Windows NT and Windows 2000 operating systems.

b. The IHS 2.0 plug-in necessary to support IHS dynamic caching was not available in IBM WebSphere Application Server Network Deployment V5.1 at the time of writing this book.

c. IHS 2.0 provides a module called mod\_mem\_cache, which supports a memory cache and a file descriptor cache.

Despite the fact that we did not focus on static content caching, Table 14-2 provides an overview of static content caching technologies:

*Table 14-2 Static caching*

	Middleware	Memory or file system
Fast Response Cache Accelerator	IBM HTTP Server	V1.3 - Memory V2.0 - Both
Web server plug-in	WebSphere Application Server - ESI processor	Memory
Caching Proxy	WebSphere Edge Components	Both



## 14.6 Benchmarking Trade3

The Trade3 application was used in various tests to demonstrate how caching can improve system performance. We include the result of two such tests here.

### 14.6.1 Dynamic caching

**Note:** This section is extracted from the existing testing scenario for dynamic caching of Trade3.

During this test the system was stressed in these four scenarios:

- ▶ No caching (see Figure 14-37 on page 586).
- ▶ Enabled command and EJB result caching (see Figure 14-38 on page 586).
- ▶ Enabled servlet and JSP result caching (see Figure 14-39 on page 587).
- ▶ Enabled Edge Side Include caching (see Figure 14-39 on page 587).

The test targeted both read and update operations as follows:

- ▶ Read (75%)
  - quote
  - portfolio
  - home
  - account
- ▶ Update (25%)
  - buy
  - sell
  - login/logout
  - register

As you can see in these figures, the command/EJB caching increased the performance 2.7 times, plus Servlet/JSP caching = 2.9 times and when edge caching was enabled, the performance was increased 4 times.

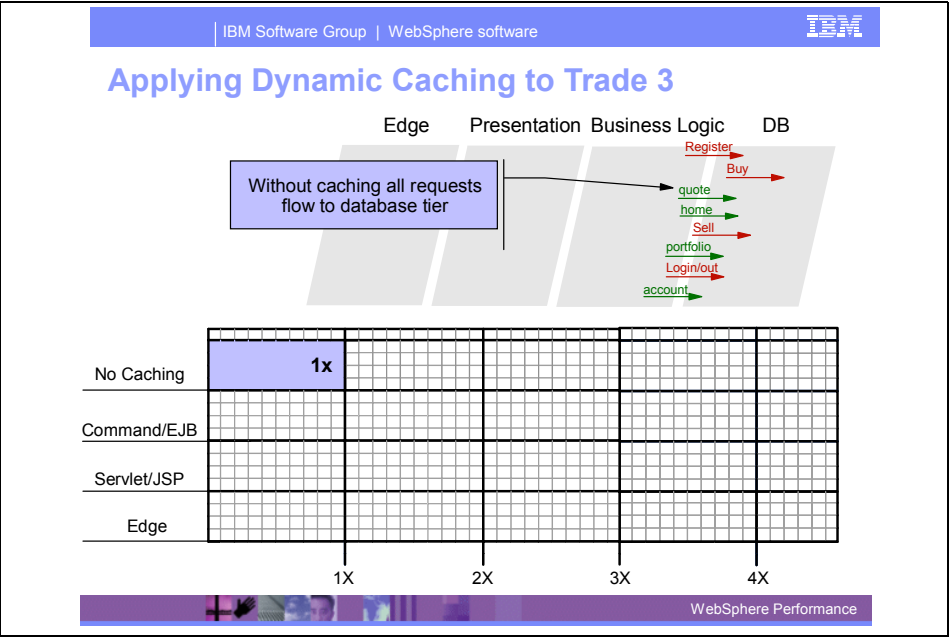


Figure 14-37 System performance without caching

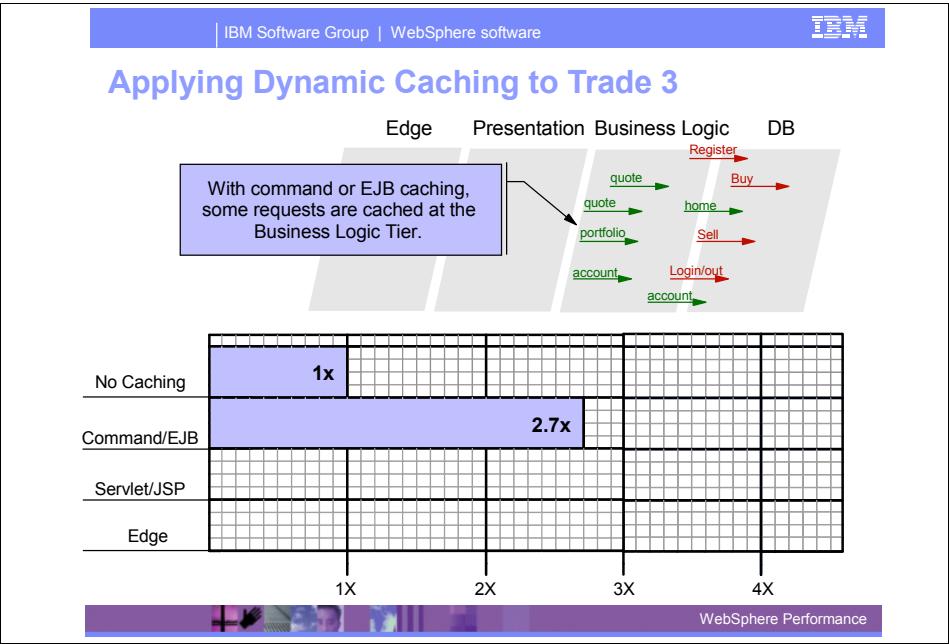


Figure 14-38 System performance with command and EJB caching

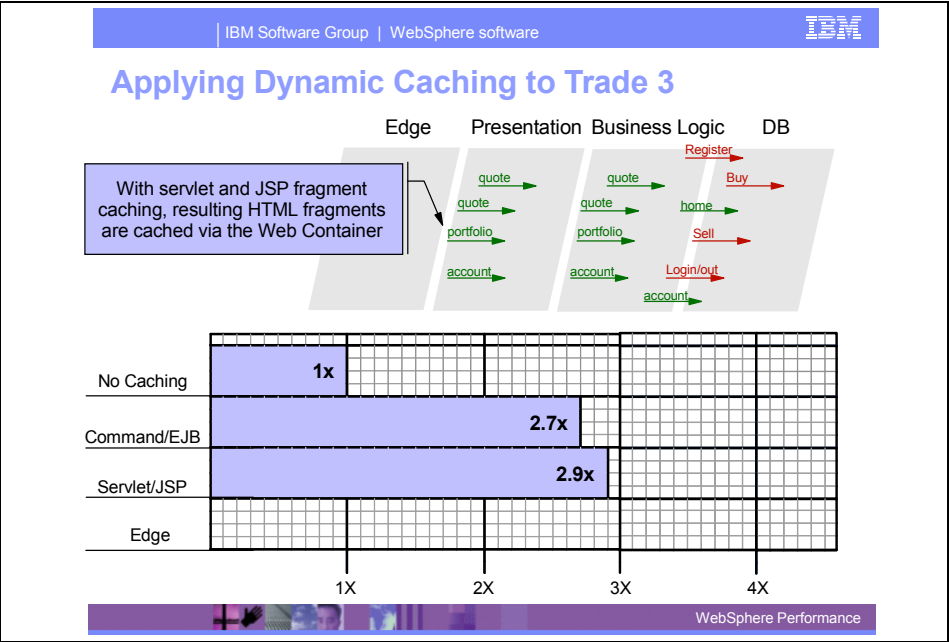


Figure 14-39 System performance with servlet and JSP caching

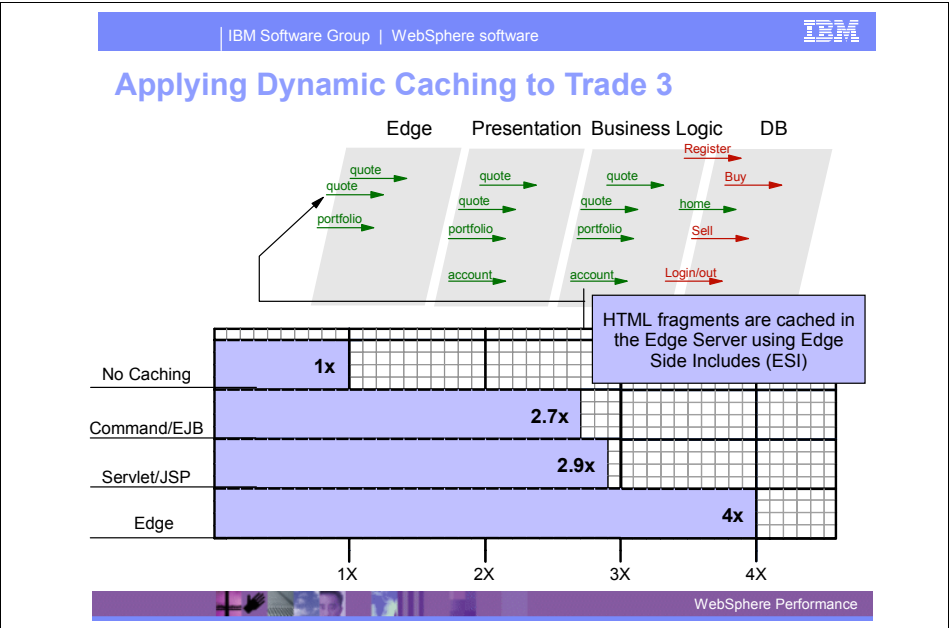


Figure 14-40 System performance with ESI processor caching

## 14.6.2 Edge Side Includes

**Note:** This section is extracted from the existing WebSphere Application Server Network Deployment 5.0 performance test. It discusses ESI performance considerations and a performance comparison for Trade3.

### Performance

First, a threaded Web server is preferred over a strictly process-based Web server for two reasons:

1. Since there is a separate instance of the ESI processor's cache for each process, a threaded Web server with fewer processes allows a higher degree of cache sharing, and thus a higher cache hit ratio, lower memory consumption, and increased performance.
2. If `ESIInvalidationMonitor` is set to true (that is, if invalidations flow back to the ESI processor), then a long-running connection is maintained from each process to the `ESIInvalidatorServlet` in each application server cluster. Furthermore, each of these connections uses a thread in the application server to manage the connection. Therefore, a threaded Web server uses far fewer connections to the back-end application server, and far fewer threads to manage these connections.

**Warning:** It was noted from our testing of Trade3 on Red Hat Linux Advanced Server 2.1 that more than 100 threads per process had adverse affects. In particular, the cost of using `pthread_mutex` locks (as is used by the ESI processor) with a large number of threads introduced a CPU bottleneck. By configuring IHS 2.0 to have 100 threads per process, `pthread_mutex` contention was minimized and the CPU utilization was maximized. The optimal thread-to-process ratio may differ for your application and will depend upon the cost of computing the data to be cached.

Figure 14-41 on page 589 shows a performance comparison of Trade3 on Windows with:

- ▶ No caching
- ▶ Servlet and command caching, but no ESI caching
- ▶ Servlet, command, and ESI caching

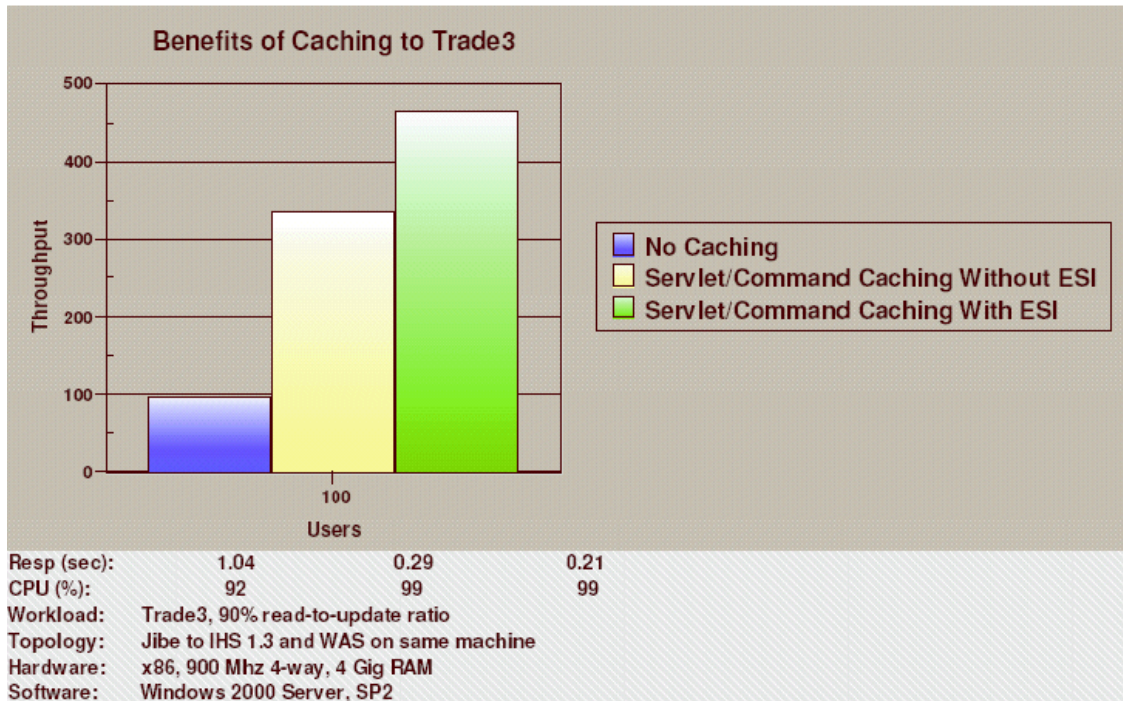


Figure 14-41 Benefits of caching

Figure 14-41 shows that servlet and command caching boost performance approximately 250%. ESI boosts the performance another 38% with a 90% read-to-update ratio for Trade3.

## Best Practices

This section contains a set of best practices for making your application's fragments "edge cacheable" (that is, cacheable on the "edge" by the ESI processor in the WebSphere Web server plug-in).

- Modify the MVC (Model-View-Controller) design practices to isolate work into individual fragments. For example, the Trade3 home page provides personalized account information for an individual user as well as generic market summary data. To efficiently edge cache this output, the page is broken into two fragments. The personalized data is in `tradeHome.jsp` which is cached for each individual user, whereas the market summary page is cached as a single instance and shared by all users. To take advantage of edge caching, the work to produce the data for each page must be done in the page itself verses in a single controller servlet which does all of the work.

- ▶ Further fragment pages when necessary. For example, the Trade3 quotes page provides quotes for an arbitrary list of stocks entered by the user. The page provides an HTML table with each row containing the price and other information for an individual stock. To take advantage of edge caching, a single JSP fragment page is created for the rows in the Quote.jsp page. This new page, displayQuote.jsp, computes the current information for a given stock symbol. This allows all stocks to be individually cached at the edge. The ESI processor will assemble the page fragments corresponding to each individual stock row to create the full table.
- ▶ Unlike the dynamic cache service which runs within the WebSphere application server, the ESI processor does not have access to user HTTP session data to uniquely identify page fragments. The application must be designed such that page fragments can be uniquely identified using request parameters on the URL, HTTP form data or HTTP cookies in the request. In a JSP include, parameters should be included in the URL as query parameters instead of as JSP parameter tags, thus allowing the ESI processor visibility to these values as query parameters.
- ▶ Consideration should be given as to how expensive a given fragment is to compute. Fragments which are expensive to compute provide the best candidates for edge caching and can provide significant performance benefits. The cache entry sizes for Dynamic Caching and the ESI processor should be large enough to accommodate these expensive fragments. Also, the priority (or the time-to-live value) of these fragments should be raised to ensure less expensive fragments are removed from the cache first.
- ▶ Another important consideration for edge caching is the update rate of a page. Invalidation of cached fragments is a relatively expensive operation. Therefore, very dynamic fragments which are invalidated often may not benefit from caching, and may actually hurt performance. Most Web application pages, however, are quasi-static and thus can benefit greatly from Dynamic Caching and edge caching in WebSphere.

## 14.7 Reference

For additional information, see the following:

- ▶ Exploiting Dynamic Caching in WAS 5.0, Part 1  
<http://www.e-promag.com/eparchive//index.cfm?fuseaction=viewarticle&ContentID=3623&publicationid=19&PageView=Search&channel=2>
- ▶ Exploiting Dynamic Caching in WAS 5.0, Part 2  
<http://www.e-promag.com/eparchive//index.cfm?fuseaction=viewarticle&ContentID=4409&publicationid=19&PageView=Search&channel=2>



## Understanding and optimizing use of JMS components

The Java Messaging Service (JMS) has become very important in developing J2EE applications. It is being used for synchronous and asynchronous calls to back end applications and also as a communication mechanism between two parts of the same J2EE application. The parts of an application that rely on JMS need to have the correct underlying messaging topology and each component needs to be configured for optimal performance and stability.

This chapter describes how the various components for JMS interact, including such areas as connection and session pools. It will take the reader through manually configuring the JMS components for the Trade3 application and demonstrate running this on both an embedded JMS architecture and also with WebSphere MQ and WebSphere Business Integration Event broker.

## 15.1 Introduction

Performance and stability of an application using JMS are largely governed by:

- ▶ Efficient and safe application usage of JMS
- ▶ The most efficient messaging and application server topology, but also one that provides adequate failure handling
- ▶ Optimal configuration settings of each individual application server and its resources

Stability is also important for maintaining performance, the application will need to be able to cope with failures in a graceful manner. If it is unable to do this then performance will degrade and the service provided by the application will become unacceptable.

It is the aim of this chapter to provide the reader with enough information to cover these areas, either through discussion or by identifying other documents that already cover a particular area.

This chapter is structured so that each section builds on the knowledge from previous sections. It starts with taking a look at what happens when you use JMS in your application. This is so that the following sections can use this to help build the larger picture of the implications of making configuration changes within a JMS setup.

Whilst some terminology and concepts are described within this chapter in completeness, others might require a previous understanding of the technology behind JMS. If you are new to JMS, then you should first look at chapters 5 and 15 in the *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195.

Also, the Trade3 application will be used for real world examples, so familiarity with this application will allow you to better understand what is being described. For more information go to:

<http://www.ibm.com/software/webservers/appserv/performance.html>

## 15.2 Components used in a JMS configuration

This first section will enable you to understand exactly what components are used when using JMS to access a messaging system. This does not cover architectures for using JMS within applications. To find out more about this take a look at:

<http://www.theserverside.com/articles/article.tss?l=JMSArchitecture>



When an application is written to access an asynchronous messaging system through JMS, it is typically in one of two ways:

- ▶ Generic JMS usage

Using the JMS API to send and receive messages. The application relies on WebSphere Application Server to locate the relevant messaging system and to handle the connections to it. What the application does with sending and receiving the messages is entirely down to the application developer within the bounds of the JMS API.

- ▶ JMS and Message-Driven Beans (MDB)

The application relies on WebSphere Application Server to monitor the underlying messaging system and pass any new messages to an instance of a purpose built MDB that will handle the message.

## 15.2.1 Generic JMS usage

Putting a message onto a message queue using JMS requires a number of components. Figure 15-1 on page 594 depicts how the PingServletToMDBQueue servlet places a message on a queue. (The complete source code for this Servlet can be found in the Trade3.ear file that comes with the Trade3 application.)

For this example the following are needed:

- ▶ JNDI Name service

Described in detail in chapter 4 of *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195.

- ▶ Queue Connection Factory (QCF)

Encapsulates the settings necessary to connect to a queue-based messaging system.

- ▶ Queue destination

A reference to the point-to-point messaging queue.

- ▶ Queue manager

A queue manager is a WebSphere MQ term. It refers to the component that is responsible for looking after a number of queues within one install of WebSphere MQ server.

- ▶ A message queue

The actual queue where messages are held until removed by an application.

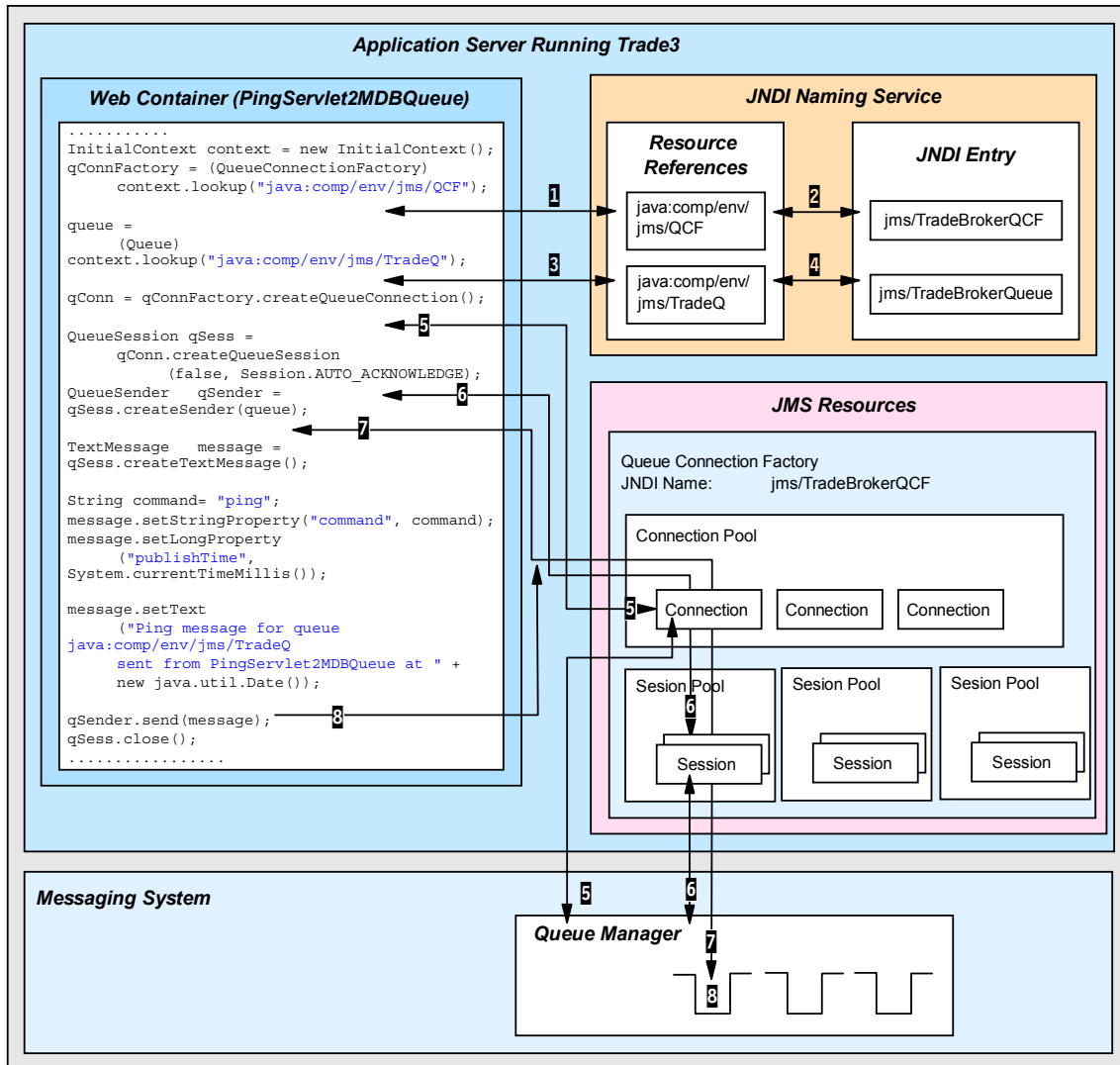


Figure 15-1 Component interactions to send a message using JMS

1. The servlet is invoked by the user and the doGet method is called by the Web container.
2. The application needs to locate the correct queue to place the message on. The JNDI namespace is used to house a link to the Java objects that will perform communication with the messaging system. The PingServletToMDBQueue servlet uses *resource references* within the code. These are linked at application deployment time to the JNDI entries that can be used to communicate with the messaging system.

3. The code performs a JNDI lookup (1) for the queue connection factory using the resource reference.
4. The resource reference is matched to the JNDI entry that contains the queue connection factory(2). The queue connection factory (QCF) object is returned, it will be used to get a connection to the queue manager that is responsible for the target queue.
5. A second JNDI lookup is performed (3) and resolved to the correct JNDI entry (4). This time it is for the queue destination, this will be used to locate the required queue from the queue manager.
6. To be able to communicate with the queue in the messaging system, the application must first create a QueueConnection (5) from the QCF object. WebSphere Application Server maintains a pool of QueueConnection objects per QCF for use by any application in the application server. This request to create a new QueueConnection will obtain an existing QueueConnection from this pool if there are any available, otherwise one will be created and added to the pool.
7. When using the Embedded JMS server or WebSphere MQ as the underlying messaging system, the creation of the QueueConnection will attempt a physical connection with the queue manager (5) to verify that it is available.
8. The next step is to create a QueueSession object using the QueueConnection method - createQueueSession (6). This is the stage where a physical connection will be established to the queue manager which will eventually be used to transmit the message to the queue. WebSphere Application Server maintains a pool of QueueSession objects for each established QueueConnection on the QCF. This request to create a new QueueSession will obtain an existing QueueSession from this pool if there are any available, otherwise one will be created and added to the pool.
9. With a connection now established to the messaging system through the QueueSession object, the application now specifies what sort of action is going to be performed, in this case it is to send a message(7). The queue definition that was taken from the JNDI name service is passed to the QueueSession object and this tells the queue manager the specific queue on to which the message will be placed.
10. The message is constructed and sent to the message queue(8).

This is just a basic example of how JMS might be used within an application. However it demonstrates the number of components that are used just to send a message. The configuration of these components can be crucial in making sure that the application performs fast enough for requirements.

## 15.2.2 JMS and Message Driven Beans

Using message driven beans incorporates further components as the application server is now responsible for delivering the message to the application.

The PingServletToMDBQueue servlet places a message on a queue associated to the JNDI name `java:comp/env/jms/TradeBrokerQueue`. Within the default configuration that comes with Trade3, a listener port has been defined that monitors this queue. Any messages that arrive on the queue are passed to the TradeBrokerMDB that then processes them. This process is shown in Figure 15-2 on page 597.

For this example the following are used:

- ▶ The components from “Generic JMS usage” on page 593:

- JNDI Name service
- Queue connection factory (QCF)
- Queue destination
- Queue manager
- A message queue

- ▶ Message listener service

The process within each WebSphere Application Server that is responsible for all its listener ports.

- ▶ Listener port

A listener port is WebSphere Application Server’s method of linking an MDB with the correct connection factory and destination. The information in a listener port definition will be used to monitor a message queue. The JMS provider will then pass any messages that arrive on the queue to the MDB that is associated to the listener port.

- ▶ Message Driven Bean (MDB)

When a message arrives on a queue that is being monitored by a listener port, the `onMessage` method of its associated MDB is called and the MDB will consume that message.

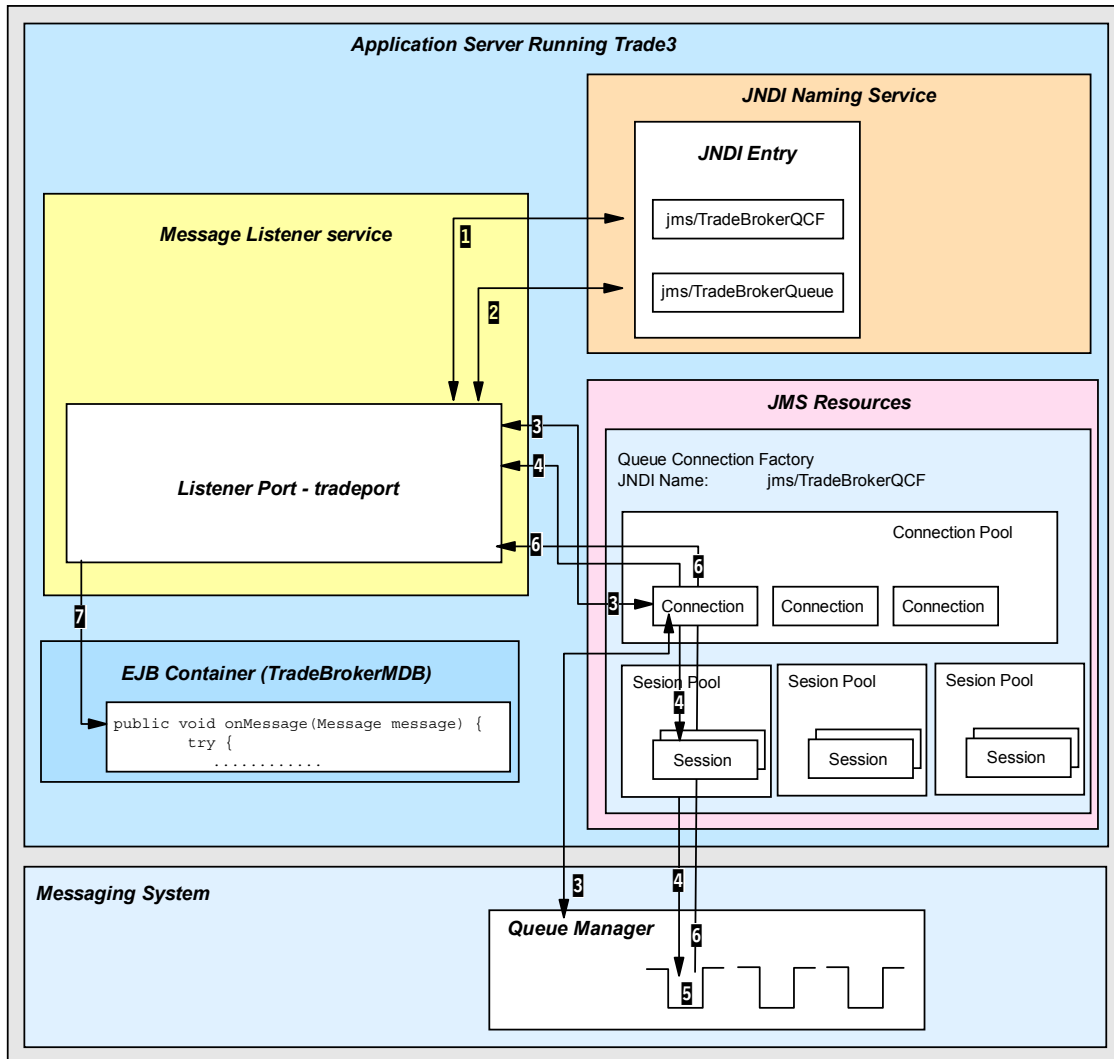


Figure 15-2 Component interactions to receive a message from a queue using a Message Driven Bean

To fully understand what is happening here, this example needs to be taken from the initialisation point of the listener port, rather than when a message arrives. A listener port can be started and stopped independently of the application server that it is running on, so this process has specifically to do with the initialisation of the listener port rather than the initialisation of the application server. By default the *initial state* of a listener port is started.

Also, Figure 15-2 does not show the detail of what is occurring inside of the message listener service. There can be multiple threads of work running inside

of the message listener service but this is not depicted, for this example assume that there is one message listener service thread that is being used by the listener port: TradePort.

1. The MDB is associated to a listener port that is in turn associated to the correct queue. Upon initialisation the listener port performs a JNDI lookup (1) for the QCF that is specified in the listener port configuration. The QCF object is returned, it will be used to get a connection to the queue manager that is responsible for the target queue.
2. The listener port will also do a JNDI lookup for the queue destination (2) that it has been configured with. This will be used to locate the required queue from the queue manager.
3. Using the QCF, the listener port connects to the queue manager through a QueueConnection (3). WebSphere Application Server maintains a pool of QueueConnection objects per QCF for use by any application in the application server. This request to create a new QueueConnection will obtain an existing QueueConnection from this pool if there are any available, otherwise one will be created and added to the pool.

Whilst the listener port is running, the QueueConnection it is using will not be available to any other application or listener port running with the application server. It is only when the listener port is stopped that the QueueConnection it was using will be returned to the connection pool on the QCF. See “Managing workload for JMS through configuration” on page 599 to understand more about the implications of this. This is also discussed in “Relationships between JMS components” on page 605.

4. The listener port now uses the QueueConnection it has created to create a ConnectionConsumer object. It is the JMS provider that actually does the listening for messages so the JMS provider uses the ConnectionConsumer to ask for listener port sessions to handle incoming messages. A listener port session can be thought of as one running thread of the listener port. It is the listener port session that calls the MDB passing it the message. Upon startup of the listener port, one listener port session will be run. At the point a listener port session is created, a QueueSession (4) is established to point at the queue the listener port is responsible for.

For ease of understanding there is only one listener port session and therefore one QueueSession object in use in this example. For more information about changing the maximum sessions see “Managing workload for JMS through configuration” on page 599 and “Relationships between JMS components” on page 605.

This is the stage where a physical connection will be established to the queue manager which is used to monitor for the arrival of messages on the queue. WebSphere Application Server maintains a pool of QueueSession objects for each established QueueConnection. Each request to create a new

QueueSession will obtain an existing QueueSession from this pool if there are any available, otherwise one will be created and added to the pool.

5. The QueueSession is now established and is pointing at the correct queue. The listener port is initialized. Any messages arriving on the message queue will be picked up one at a time and processed by the correct MDB.
6. A message arrives (5) on the queue.
7. As the listener port is ready to accept messages, it will take delivery of the message off the queue (6). The listener port will then pass the message on to an instance of the MDB that it is associated with. The *onMessage* method is called on the MDB (7) passing it the message.

Using MDBs requires an understanding of many different areas. The application can be designed to use MDBs, but the way in which those MDBs behave can be significantly affected by the way in which its controlling components are configured - the message listener service and listener ports. Those are in turn affected by the underlying communication mechanisms to reach the messaging system, for example the number of QueueConnections and QueueSessions available on a QCF.

## 15.3 Managing workload for JMS through configuration

In 15.2, “Components used in a JMS configuration” on page 592 we saw how the JMS components interact. This only showed a single request, where in reality there would be many simultaneous requests. A Java application server’s ability to handle multiple *threads* of work allows an application to perform well when the workload requires dealing with simultaneous requests. In a Web application, simultaneous user requests will be serviced by Web container threads. If the Web application makes use of JMS, then the JMS components will also be required to handle simultaneous requests.

For applications which have MDBs, the workload also arrives via the messaging system. In this situation it might also be optimal to have messages processed in parallel.

All the workload that is arriving to do with JMS, either from MDBs or from other requests, needs to be efficiently handled within the bounds of the physical resources available to the application server and the underlying messaging system (and also any back end applications accessed through messaging).

There are settings within each of the JMS components that allow controls to be placed on that workload to create a balance that provides the best throughput and response times for the physical hardware the application is running on.

This section describes the configuration settings that can often have the most influence on performance. More information to help choose the optimal settings for these values and some more settings can be found in “Choosing optimal configuration settings” on page 610.

### 15.3.1 Basic workload patterns

The way in which work arrives at the application server determines how many simultaneous JMS requests will be handled. This affects which settings will make a difference on how JMS is used.

#### Workload from Web or EJB container

When requests come from the Web and / or EJB container it is the number of threads in these containers that is controlling the upper limit for the number of simultaneous accesses to the JMS resources.

The QCF and *Topic Connection Factory* (TCF) objects control the pool of objects needed to access the underlying messaging system and they also manage that access.

#### Key values

- ▶ Maximum sessions in session pool on QCF or TCF
- ▶ Maximum connections in connection pool on QCF or TCF

#### Key point

- ▶ The number of QueueSessions and QueueConnections is dependent on the number of requests from Web and EJB containers and on how the application has been written to use these objects.

Figure 15-3 on page 601 shows a typical way in which requests might flow in to an application. In this example all access to the QCF is through the EJB container and messages are only being placed on the queue. On each set of requests denoted by the arrows, is a tag that represents the controlling factor on how many simultaneous requests can reach that particular resource.

As requests arrive, a proportion of them require use of JMS resources. There has to be enough of these resources available to handle all the requests. If there are not enough JMS resources available then requests will have to wait for them to become free. This could lead to a time out and requests having to be re-submitted.

Ideally, each request should only ever require the use of one QueueConnection and one QueueSession at any time. Programming in this manner avoids a deadlock situation. However, it is possible to use more than one of each or cache



the QueueConnection and use across many threads as it is thread safe (the QueueSession object is not thread safe - one is needed for each thread). So the number of QueueConnection objects and QueueSession objects can vary both on the number of requests requiring JMS and on how many objects that request will use.

Within the definition of a QCF or TCF it is possible to change the number of QueueSession objects and QueueConnection objects that are available to the application.

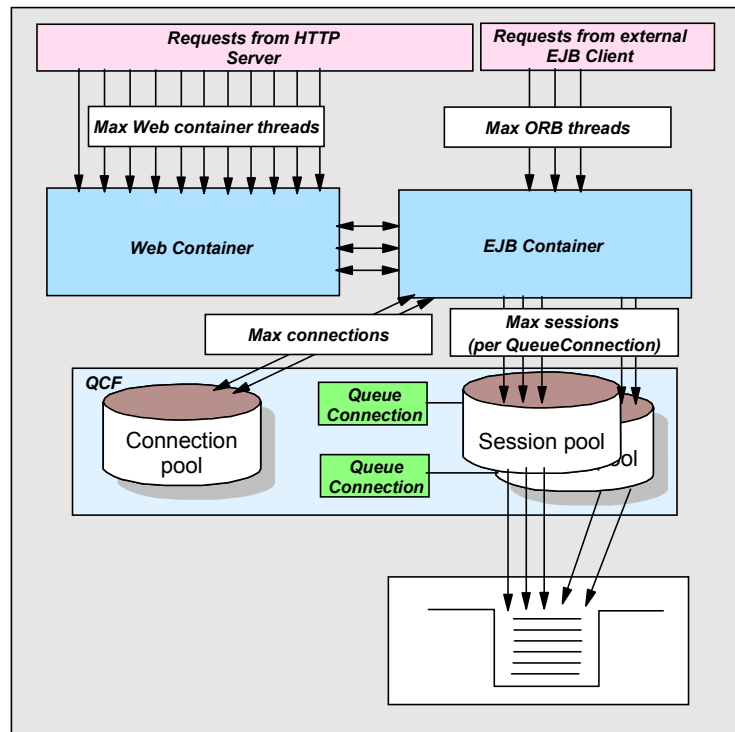


Figure 15-3 Request flow for Web and EJB containers accessing JMS resource

You can view the connection pool and session pool objects in the configuration information of the QCF or TCF definition at the correct level of scope for your application. See “Choosing optimal configuration settings” on page 610 for more information on using scope when defining JMS objects.

Node	app1
Component-managed Authentication Alias	(none)
Container-managed Authentication Alias	(none)
Mapping-Configuration Alias	DefaultPrincipalMapping
XA Enabled	<input type="checkbox"/> Enable XA
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>	
<b>Additional Properties</b>	
<a href="#">Connection Pool</a>	Connection pool settings for JMS. These settings are not used for Generic JMS Connection Factories.
<a href="#">Session Pools</a>	An optional set of session pool settings.

Figure 15-4 The connection pool and session pool settings on a QCF settings page

Within these two options it is possible to set a minimum and maximum value for how many connections and sessions are available for the application to use.

**Important:** Every QueueConnection has its own pool of QueueSession objects. This means that if the connection pool has a maximum of 5 and the session pool has a maximum of 10, providing the application is written to do this, there can be up to 50 QueueSessions open, 10 on each of the QueueConnections.

More information about setting these values and some other settings that can affect performance can be found in “Choosing optimal configuration settings” on page 610. For information on monitoring these values in real time see “Monitoring JMS performance within Tivoli Performance Viewer” on page 679.

## Workload from messaging system (MDB)

In this pattern, the workload is generated by messages arriving on the message queues that the listener port (or ports) is monitoring. This means that the amount of workload that comes into the application server is directly associated to how many active sessions are running on each listener port.

### Key values

- ▶ Maximum sessions in session pool on QCF or TCF
- ▶ Maximum connections in connection pool on QCF or TCF
- ▶ Maximum sessions on listener port
- ▶ Maximum threads on message listener service

### Key point

- Knowing the number of listener ports defined in an application server and the maximum sessions on each listener port, allows you to work out the other settings.

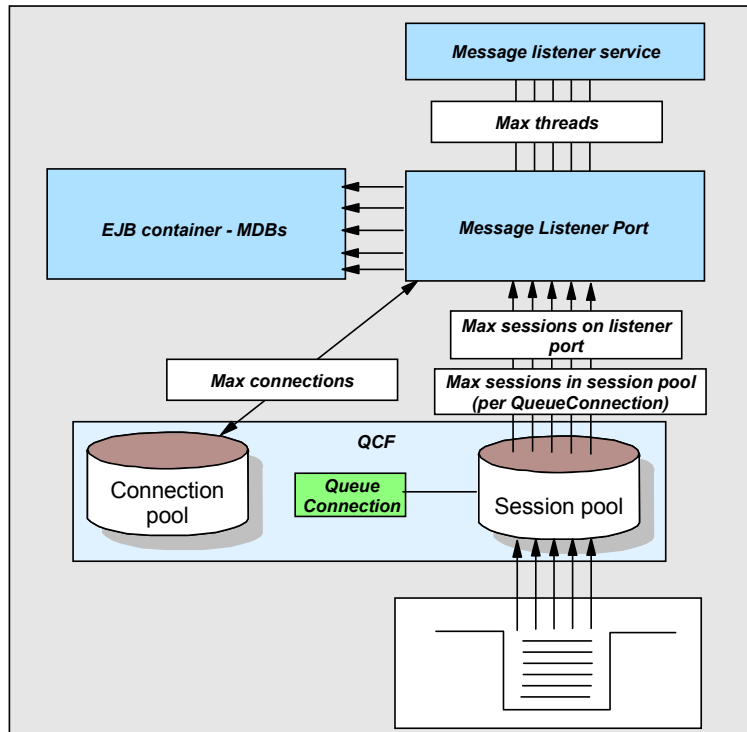


Figure 15-5 Request flow when using MDBs

If the maximum sessions for a listener port is set to 5, then there can be up to five sessions working in parallel to process messages on the message queue. The default for this value is 1 which would in effect make messages be processed in a serial fashion. Setting it to any more than 1 means that the order that messages get processed in cannot be guaranteed.

**Explanation:** 3 listener port sessions are working off one message queue with 3 messages on it, each session picks up a message. There is a chance that session 2 could finish first, before sessions 1 and 3 have completed. This would mean message 2 is processed before message 1.

Transaction rollback would also cause messages to be processed in a different order. Same scenario, but this time message 2 is rolled back. It is put back on the queue, and so could end up being processed after message 3 has completed. If you have Maximum sessions set to 1, then message 3 won't be processed until message 2 has been successfully processed.

As this setting governs how much simultaneous activity will occur in the application server it also governs how much of the other resources will be needed. When a listener port session is in use it needs:

- ▶ One message listener service thread
- ▶ One QueueConnection (a listener port only ever uses one QueueConnection regardless of the number of listener port sessions)
- ▶ One QueueSession

Choosing the correct number of listener port sessions to handle the messages as they arrive defines all the other parameters.

Ideally, the number of sessions on a listener port must be sufficient to handle the peak load of messages arriving on the message queue. This would have to be balanced against the system resources and all the other processing that is going on in the application server.

**Important:** A listener port will use one QueueConnection object regardless of the number of listener port sessions, and that object will remain in use as long as the listener port is started.

A listener port session, once established will hold on to all the resources it needs until it is finished. This means it will be using up a QueueSession and a message listener service thread.

For more information about understanding the associations between the settings listed above see “Relationships between JMS components” on page 605. To learn more about these settings and others go to “Choosing optimal configuration settings” on page 610.

Finally for information on monitoring usage of these settings go to “Monitoring JMS performance within Tivoli Performance Viewer” on page 679.

## **Workload from both**

If workload is arriving via the Web and EJB containers and also via MDBs then there has to be enough resources available to cope with both. Unless specified, both sets of workload will use the same objects from within WebSphere Application Server. This means that requests for sending and receiving messages will need to obtain QueueConnection and QueueSession objects from the same QCF as the message listener service.

### ***Key values***

- ▶ All of the values mentioned in the previous two sections.

### ***Key point***

- ▶ When the application is receiving a workload from both directions, front end and back end, it is essential to correctly configure the QCF and TCF objects to avoid unnecessary waiting.

As the amount of resources to run the message listener service can be calculated based on the maximum number listener port sessions, it might be worth having two QCFs or TCFs. In this way it is possible to split the workload back in to a QCF for Web and EJB container workload and a QCF for the message listener service.

Configuring QCFs and TCFs to split the workload is covered in “Choosing optimal configuration settings” on page 610.

## **15.4 Relationships between JMS components**

The more locations in an application that use JMS, the harder it is to optimally configure all the JMS components. There are many relationships and dependencies between the components, some of which have already been discussed above. When configuring a component in the JMS infrastructure it is essential that you understand what resources will be needed to make it work. When changing certain parameters the requirements the change will place on the rest of the components is not always obvious.

### **15.4.1 Component relationships when using MDBs**

This is especially the case when using MDBs. Figure 15-6 on page 606 shows how all the JMS components are linked together to allow an MDB to work when receiving point-to-point messages via a QCF. The relationships would be the same if it was for a TCF. This can be used to help map out the impact of making changes to the configuration.

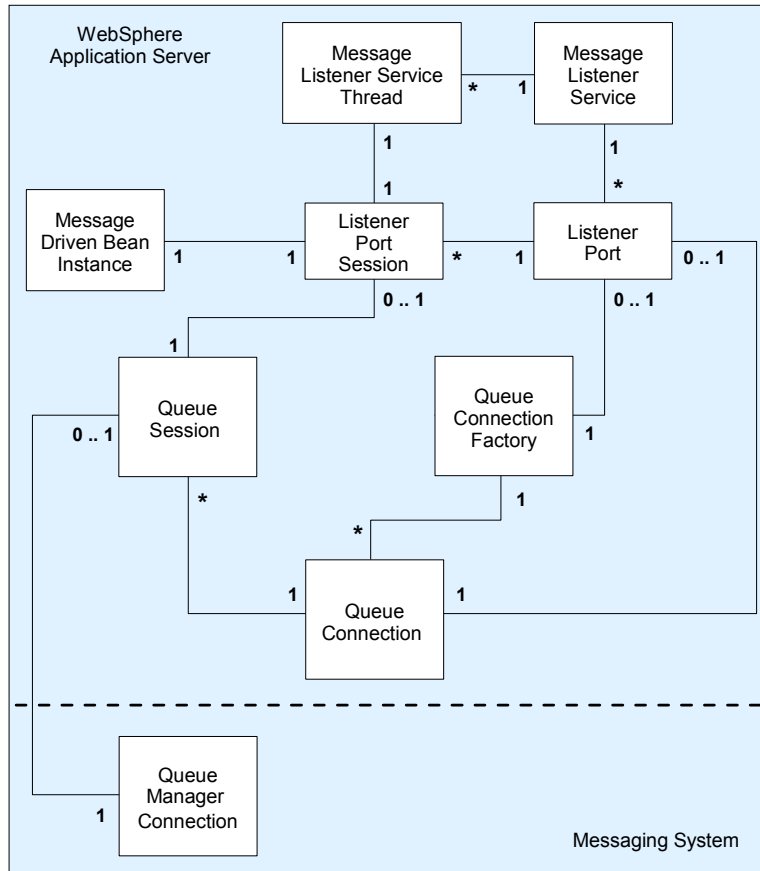


Figure 15-6 Relationships between JMS components for MDB

To read this diagram, select the component you want to understand more about and then read the connectors going from it to the other components. For example 1 QueueSession object can have 0 or 1 Listener port sessions associated to it, it will also have 1 physical queue manager connection associated to it.

Table 15-1 on page 607 and Table 15-2 on page 608 will help in interpreting Figure 15-6. Table 15-1 on page 607 describes how to interpret individual relationships.

Table 15-1 Diagram Key for Figure 15-6 on page 606

Relationship	Meaning
1 to 1	Where the relationship is 1 to 1 this means that for the first component to operate there has to be one of the second component available. <b>Example:</b> A listener port session requires a message listener thread. A message listener thread is always associated to a listener port session.
1 to 0 .. 1	Where the relationship is 1 to 0 .. 1 this means that the first component can <i>optionally</i> be associated to the second component. <b>Example:</b> One QueueSession can have 0 or 1 listener port sessions associated to it.
0 .. 1 to 1	Reads the same as 1 to 1. See 1 to 1.
1 to *	1 to * means that component one has many instances of component two associated to it. On this diagram a 1 to * is showing control. Component one is in charge of multiple component two's. <b>Example:</b> The message listener service has many message listener threads.
* to 1	Reads the same as 1 to 1, except that when it is a * to 1 relationship it showing a child - parent relationship. <b>Example:</b> One QueueSession is managed by one QueueConnection.

Some of the boxes on the diagram represent configurable components that can be altered, changing the number of them that exist. These relationships become important when configuration changes are made as each change could have a knock on effect. Table 15-2 on page 608 shows some examples of using Figure 15-6 on page 606 to help in finding those side effects.

Table 15-2 Example linked relationships for JMS components

Scenario	Requirements
Increase the number of QueueSessions by one.	As there is only a single 1 to 1 relationship for a QueueSession the only requirements to be able to do this are to make sure there are enough queue manager connections available. Remember though, by increasing the number of QueueSessions in the session pool by one, it is affecting all the session pools for all the QueueConnections established from that QCF. So it will in fact allow QueueConnections * 1 more sessions, not just one more session. The setting for the max sessions in the session pool will be determined by the QueueConnection that needs the most sessions.
Increase listener port sessions by one.	Reading the diagram shows that a listener port session is reliant on a QueueSession and a message listener thread. This means that there needs to be enough of these available for it to be increased. Increasing the message listener threads has no impact on any other component, except that there need to be enough message listener threads for <i>all</i> listener port's sessions. Increasing the QueueSessions is described above.
Add a new listener port with one listener port session.	To do this there needs to be enough message listener threads, QueueConnections and QueueSessions available. Increasing the number of QueueSessions will not be necessary as this new QueueConnection has its own pool of QueueSessions, which has at least one in the pool.

## 15.4.2 Component relationships when using generic JMS

The relationships are not as complex when using JMS to send and receive messages within your application code. This is much more dependent on how you code your application.

As was discussed in “Basic workload patterns” on page 600 for Web and EJB container workloads, doing generic JMS calls means using QueueConnections and QueueSessions, whether it is work with publish and subscribe or point-to-point.

As was mentioned earlier, QueueConnection objects are thread safe so they can be cached and then used across multiple threads in the application. However, QueueSessions are not thread safe so each thread will need its own QueueSession. This means that at a basic level your application will probably fall into either Figure 15-7 on page 609 or Figure 15-8 on page 609.



These diagrams represent a single point in time. For example, in your application thread it could be using more than one QueueSession, but it would only be using one QueueSession at a time. These diagrams represent two extremes of using JMS objects. There are variations in between, for example the QueueConnection could be cached at a lower level than across the whole application, doing this would again affect the relationship between the JMS components and the application.

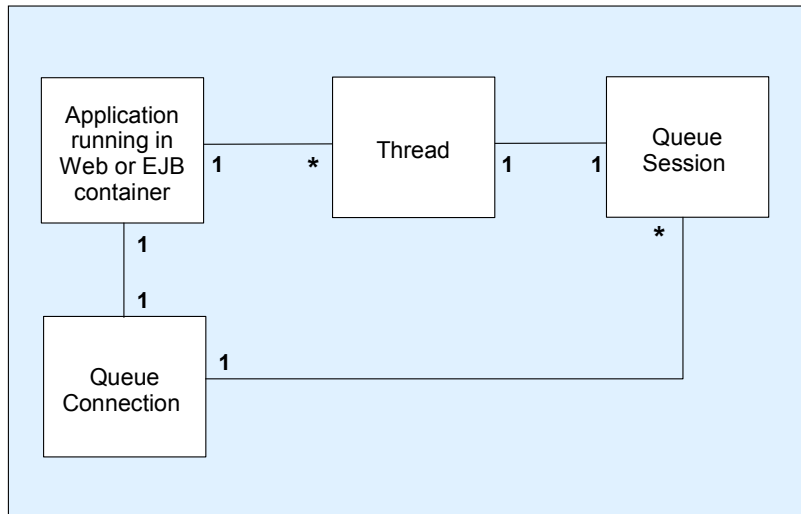


Figure 15-7 JMS component relationships when caching QueueConnection

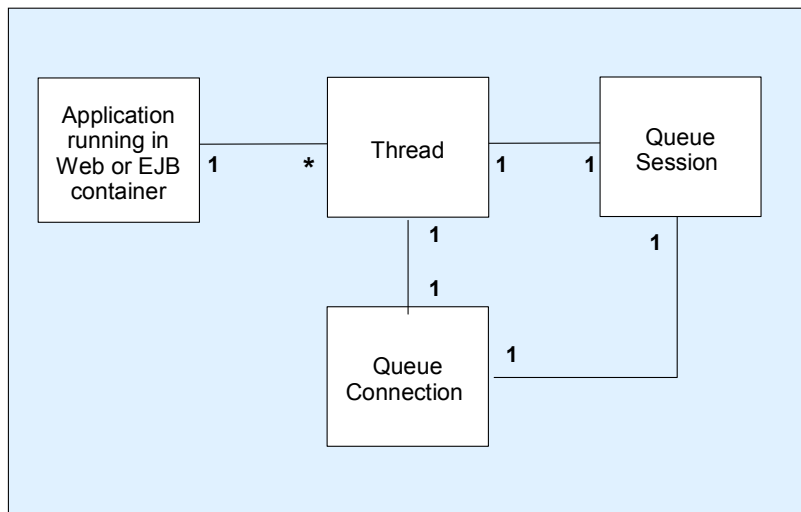


Figure 15-8 JMS component relationships when not caching the QueueConnections

To understand how the JMS components are related requires an understanding of how the application is using those components. Unlike when using the message listener service, it is the application design or development that will define how these are used and therefore what the relationship is between the components.

For a QCF or TCF, configuring the number of QueueConnections in the connection pool, and QueueSessions in the session pool ultimately relies on knowing the way in which the code is written.

Caching QueueConnections means that more QueueSessions are needed in the session pool.

### ***Example 1***

There are 30 application threads all needing to simultaneously place a message on a Queue through the same QCF. One QueueConnection is obtained and shared across the threads. This then means that 30 QueueSessions are needed from that one QueueConnection. Connection pool maximum only needs to be 1 in this example, session pool maximum size needs to be 30.

Using a new QueueConnection for each thread means that there will probably be a 1 to 1 relationship between the QueueConnection and QueueSession used on that thread, so there will need to be more QueueConnections in the connection pool and less QueueSessions.

### ***Example 2***

There are 30 application threads all needing to place a message on a Queue through the same QCF simultaneously. Each thread obtains a QueueConnection and from that a QueueSession. This means that 30 QueueConnections are needed and only one QueueSession per connection. Connection pool max would need to be 30 but the session pool maximum would only need to be 1.

These are two extreme scenarios and the values in the examples do not take into account any other part of the application.

## **15.5 Choosing optimal configuration settings**

This section is a list of guidelines and settings that can help in finding the all important starting point for performance tuning an application. It is intended to help add some reason in choosing specific settings.

**Important:** Each application will operate differently in the way it uses the JMS components. Any values mentioned in this section should not be treated as a generic optimal value.

Unless stated, the settings listed in this section apply to both the WebSphere JMS provider (Embedded JMS server) and WebSphere MQ JMS provider (WebSphere MQ).

### 15.5.1 Creation of the JMS provider objects at the correct scope

If your WebSphere Application Server application is going to be clustered across multiple application servers then the messaging system has to be set up so that message sending and receiving works correctly in that cluster.

There are many different topologies that can be configured for the underlying messaging system. A detailed discussion on these topologies can be found at:

[http://www.ibm.com/developerworks/websphere/library/techarticles/0310\\_barci\\_a/barcia.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0310_barci_a/barcia.html)

This paper discusses some of the choices available to the infrastructure designer in choosing the optimal topology for an application to use. “Example JMS topologies and scenarios” on page 639 demonstrates two topologies that could be used in a production environment and some reasons for choosing them.

Before going into configuring pool sizes and other QCF or TCF settings it is important that the scope level under the desired JMS provider is correct. Within WebSphere Application Server V5.1 there are three levels of scope when defining a resource that affect its visibility to applications:

- ▶ Cell

If a resource is defined at cell level then it will be visible to all nodes and application servers within that cell.

- ▶ Node

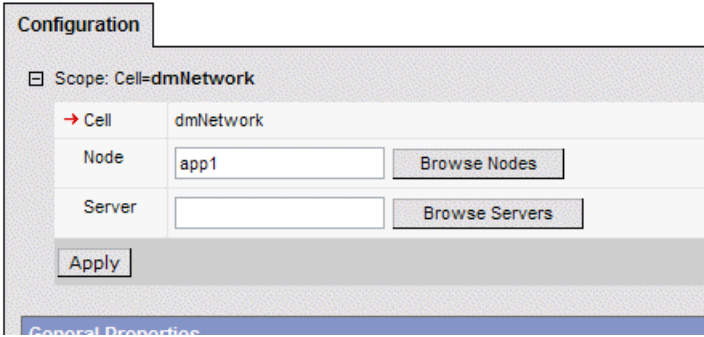
A resource defined at node level is only visible to that node and any application servers residing on that node.

- ▶ Server

Resources at server level are only visible to that application server and nowhere else.

## WebSphere JMS Provider

A JMS provider enables asynchronous messaging based on the Java Messaging Service objects are used to manage JMS resources for the internal WebSphere JMS provider. [i](#)



The image shows a 'Configuration' dialog box for the WebSphere JMS Provider. It has a tabbed interface with 'Configuration' selected. The 'Scope' is set to 'Cell=dmNetwork'. Below this, there are three rows: 'Cell' with a dropdown arrow and the value 'dmNetwork'; 'Node' with a text box containing 'app1' and a 'Browse Nodes' button; and 'Server' with an empty text box and a 'Browse Servers' button. At the bottom left is an 'Apply' button. A 'General Properties' tab is visible at the bottom.

Configuration	
☐ Scope: Cell=dmNetwork	
→ Cell	dmNetwork
Node	app1 <span>Browse Nodes</span>
Server	<span>Browse Servers</span>
<span>Apply</span>	

Figure 15-9 The scope level setting

Depending on the underlying messaging system topology it might be necessary to use this scope level to match up application servers with the correct queues and queue managers.

Defining a QCF at the *cell* level will mean that *all* applications running in the cell will see this resource. When the QCF is defined, it is the specified JNDI name that will be used by the application to find the QCF. If the QCF is defined at the cell level then that QCF will be bound to that JNDI name in all processes started in the cell. Consequently any call to that JNDI name will link to the same QCF and therefore to the underlying messaging component that is associated to that QCF.

For example in WebSphere MQ, this would assume that any part of an application using this QCF, in any server in the WebSphere Application Server cluster, would be accessing one queue manager as is shown in Figure 15-10 on page 613.

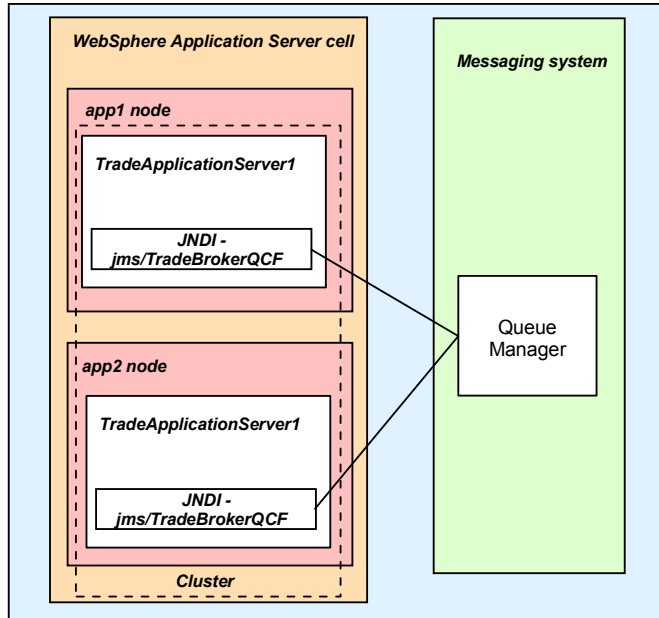


Figure 15-10 Define a resource at cell level

By defining the QCF at *node* level it would be possible to point the application servers running on app1 to one queue manager and application servers on app2 to a different queue manager. This could be done to remove the single point of failure in the one queue manager or to increase performance by having two clustered Queue managers. There are many factors that affect the choice of messaging topology and this is not a discussion about them, the point here is that the scope setting allows greater flexibility in deciding on the best topology for your application.

Setting the scope level correctly not only affects how the application interacts with the messaging system but it also defines how many QCFs and TCFs you are going to have to create and administer. If there are four nodes in your cell and the underlying topology requires that each node accesses a different queue manager, then that is going to be four QCFs created at node level and therefore four sets of connection and session pool settings to change.

More information and some examples on configuring QCFs and TCFs at different scope levels to match the messaging topology can be found in “Example JMS topologies and scenarios” on page 639.

## 15.5.2 Important JMS component settings

As well as the connection pool and session pool settings on a QCF and TCF (see “Setting up the QCF / TCF pools” on page 628) there are also a number of other settings that could affect performance. The settings will only work with the correct combination of messaging topology, JMS provider, and application usage of JMS. This is specified under each heading.

### Transport Type - BINDINGS or CLIENT

This setting determines the method the MQ JMS classes use to communicate with the queue manager.

- ▶ JMS Provider: WebSphere MQ JMS
- ▶ Component: QCF or TCF
- ▶ Applies to specific messaging topologies: Yes
- ▶ Requires specific usage of JMS in application: No
- ▶ Location of setting:
  - **Resources -> WebSphere MQ JMS Provider -> WebSphere MQ Queue Connection Factories -> <Your\_QCF> -> Transport Type field**
  - **Resources -> WebSphere MQ JMS Provider -> WebSphere MQ Topic Connection Factories -> <Your\_TCF> -> Transport Type field**

If the queue manager is running on the same physical machine as the WebSphere Application Server then it is possible to set the transport type to BINDINGS. If the queue manager is on a remote machine then CLIENT has to be used. This could also affect whether transaction support is provided to MQ. To enable XA support the queue manager must either be accessed using BINDINGS (and therefore local) or accessed using CLIENT and the specific XA enabled MQ client must be installed on the application server machine.

Using the BINDINGS transport type removes some of the overhead of remote access and is generally faster as it does not have to perform as many actions to access the queue manager.

### Persistent Messages

Message persistence means that if a queue manager or Embedded JMS server fails, all the messages it holds will be recoverable as they are written to disk.

- ▶ JMS Provider: WebSphere JMS and WebSphere MQ JMS
- ▶ Component: Queue destination and Topic destination
- ▶ Applies to specific messaging topologies: Yes
- ▶ Requires specific usage of JMS in application: Yes
- ▶ Location of setting: Varies - see text.

The configuration of whether a message should be persisted is configurable at three levels:

► Application level - In the application code

When using either the `QueueSender` or `TopicPublisher` JMS classes it is possible to specify whether messages should be persistent. This is done using the `setDeliveryMode()` method which accepts values of either:

- `DeliveryMode.NON_PERSISTENT`
- `DeliveryMode.PERSISTENT`

When creating a `QueueSender` or `TopicPublisher`, if `setDeliveryMode` is not used then the default is `DeliveryMode.PERSISTENT`.

► Queue destination and topic destination settings

Within these objects it is possible to define the persistence. Possible values for persistence are:

- `APPLICATION_DEFINED`
- `PERSISTENT`
- `NON_PERSISTENT`
- `QUEUE_DEFINED` (only on the WebSphere MQ JMS provider objects)

► Queue definition in queue manager

For WebSphere MQ it is possible to set the Default Persistence.

Even though you can configure the persistence at each level there are some rules that govern which setting will take precedence:

- Setting the value on the WebSphere MQ queue is only the default persistence setting. It is overridden by the setting on the JMS queue and topic destinations.
- Any value set in the application code will be overridden by the settings on the queue or topic destination unless they are configured as `APPLICATION_DEFINED`, in which case the value in the application will be used.
- Defining `PERSISTENT` and `NON_PERSISTENT` on the queue and topic definitions is the only way to accurately know from looking at the WebSphere Application Server configuration whether persistence will occur.
- If you are using WebSphere MQ (as opposed to Embedded JMS) then you can configure the queue or topic destination to take the default value specified on the queue. This is done using `QUEUE_DEFINED`.

As messages are being stored on disk this reduces the performance of message processing. Persistence should only be used when the topology in use demands it. For example, if you decide to use one queue manager and have it made highly

available using some HA software like HACMP, then you will need to have persistent messaging turned on to make this work.

Alternatively, if you have an architecture whereby messages are sent and received in a synchronous fashion, then it might be the case that the client code only waits for a certain amount of time before re-sending the message. In this instance the topology would contain more than one active queue manager and persistent messages would not be required as failures would be masked by retries.

It is not straight forward to decide whether persistence is needed and for this reason it is much better that it not be part of the considerations for a developer. It is recommended that persistence is not specified by the application and is left to the definition of the queue or topic destination so that it is obvious to the administrator whether messages will be persisted.

**Attention:** When a queue or topic destination is defined the default is APPLICATION DEFINED. As the default for creating a new QueueSender or TopicPublisher object in the code is to have messages persisted, the overall defaults for setting up a queue destination are that the message will be persisted. If you do not need messages to be persisted then you need to change the APPLICATION DEFINED to NON PERSISTENT on the queue or topic destination.

## **XA enabled resources**

XA support extends transaction support as it allows WebSphere Application Server to coordinate a transaction over many back end resources, committing the results to those resources only when all the work in the transaction has been completed.

- ▶ JMS Provider: WebSphere JMS and WebSphere MQ JMS
- ▶ Component: QCF and TCF
- ▶ Applies to specific messaging topologies: Yes
- ▶ Requires specific usage of JMS in application: No
- ▶ Location of setting:

**Resources -> WebSphere JMS Provider or WebSphere MQ JMS Provider  
-> WebSphere MQ Queue Connection Factories -> <Your\_QCF> ->  
Enable XA checkbox**

**Resources -> WebSphere JMS Provider or WebSphere MQ JMS Provider  
-> WebSphere MQ Topic Connection Factories -> <Your\_TCF> -> Enable  
XA checkbox**



2phase commit can only operate with XA enabled resources and this setting specifies this.

By default, XA support is enabled when creating a QCF or TCF. With XA enabled more work has to be done to complete the transaction. It is an expensive overhead if there is no use of two phase commit in your application and so should be disabled. Be aware that disabling XA support will only work if your application does not use this resource as part of a global transaction. Turning this off does not change behavior of a transaction, if XA enabled is set to false and the application tries to enlist more than one resource in the transaction, then an exception will be thrown.

If your application does need to use an XA enabled QCF or TCF then you should establish whether it is needed for *all* interactions with the messaging system. If it is only needed for a small section of the application requests then it is recommended that two connection factory objects are created, one with XA enabled and one without. This should then make the parts of the application that do not need XA run faster.

## **Client ID and Enable Clone support on a TCF**

The enable clone support setting is specifically for durable subscription based publish/subscribe messaging on the WebSphere products. The Client ID is used to aid clone support of publish/subscribe messaging. Enable clone support permits listener ports defined with the same TCF to share a durable subscription, as long as they have a cloned application deployed against them.

- ▶ Component: TCF
- ▶ Applies to specific messaging topologies: Yes
- ▶ Requires specific usage of JMS in application: No
- ▶ Location of settings:

**Resources -> WebSphere JMS Provider or WebSphere MQ JMS Provider  
-> WebSphere MQ Topic Connection Factories -> <Your\_TCF> -> Enable  
Clone Support** checkbox

**Resources -> WebSphere JMS Provider or WebSphere MQ JMS Provider  
-> WebSphere MQ Topic Connection Factories -> <Your\_TCF> -> Client  
ID** field

When using publish/subscribe there are two ways in which client subscriptions can be handled, durable and non-durable. If a client subscribes to a topic using a non-durable subscription and then is terminated or loses network connectivity, it will not receive any future publications until it is started again. Any publications occurring whilst the client is unavailable will be lost.

Durable subscriptions mean that a client's subscription lives past the termination or loss of communication of that client. Whilst the client is unavailable, the broker keeps a copy of any published messages requested in the client's durable subscription. When the client starts back up again the messages are then received by the client.

**Important:** As a durable subscription lives past the termination of the application client, the only way to remove the durable subscription is to deregister it from the broker. If you are using a durable subscription on an MDB then the durable subscription is only removed when the application is removed from WebSphere Application Server, and only then when that application server is restarted so that it can tell the broker of the subscription change.

There are different rules for how to create a durable and non-durable subscription, because of the persistence needed for a durable subscription.

A non-durable subscription works in a similar way to listening on a queue for a message to arrive. In the case of a message driven bean, WebSphere Application Server uses a listener port to create the necessary JMS objects to point to the required topic. If any of those objects are closed or the link with the message broker is lost then the subscription is also lost. From the broker's perspective it recognizes a distinct subscription by the TopicSubscriber that has been used. Each communication channel opened to the broker is a new subscription from a new client.

A durable subscription cannot be identified by the connection to the message broker as once it is closed or lost, so would be the subscription. Instead, the broker recognizes the subscription based on the ID of the durable subscription. This ID is based on a set of parameters which form a unique reference. If the client should stop, then when it is started again it will be recognizable to the broker by the fact that it has the same ID.

If a client, in particular an MDB with a durable subscription, wishes to subscribe, it needs to have a unique durable subscription ID. This ID is created using the:

- ▶ J2EE name of the MDB
- ▶ The client ID setting on the TCF assigned to that MDB's listener port

Problems start to occur if the application is clustered. If there is only one installation of the application running, then this MDB's subscription remains unique for this broker. But if another copy of the application is started up pointing at the same TCF then the subscription is no longer unique on the broker. The second MDB will not function as expected.

For this reason, when clustering an application you will need to carefully choose how many TCFs serve your MDBs and the settings on those TCFs.

Two different types of behavior with a durable subscription and MDBs are possible:

- The application is installed in a cluster. All the MDB listener ports are configured to point at one TCF, which is defined at the lowest scope that allows all your application servers to view it. This will be either cell or node depending on your topology. This TCF has **Enable clone support** checked and a **Client ID** specified (both are fields on a TCF for Embedded JMS and MQ JMS providers).

When all the servers in your cluster are started, the MDBs and their listener ports will start successfully. When a message is published *only one of the MDBs will get the message, so only one of the cluster members will receive the information within the message*. This is because the broker only sees one subscription as the client ID on the TCF and the name of the MDB are the same for each application server. It is at this point that the Enable clone support setting needs to be used. The JMS specification states that there can only be a single subscriber for a particular durable subscription at any one time. Enabling clone support allows all the MDBs to successfully start even though they all have the same client ID for the durable subscription. Without this setting turned on, only one of the MDBs will work properly. With enable clone support turned on, the JMS provider will then pass a publication to any one of the MDBs.

Whichever of the listener port sessions is available first will receive the message. This is like one grouped line of customers waiting to be served by a number of cashiers, where the line of customers is the topic publication messages and the cashiers are the listener ports. There are four cashiers working, each dealing with a customer, when one of them finishes their work the next customer in the line goes to that cashier. If a line never forms then the cashiers will not be busy and so the next customer to arrive could go to any of them. This method of distributing the workload means there is no algorithm that can be applied to understand which cluster member will get any given message.

This behavior occurs because the broker sees one subscription. When a message is published that matches this subscription, the MQ JMS classes take the message and then pass it to whichever of the listener ports is available first to process the message. This behavior occurs because Enable clone support has been specified, it is telling the MQ JMS provider to allow multiple MDBs (one in each clone), but only use one at a time.

**Note:** This scenario means that all MDBs use the same communication mechanism to reach the message broker.

- ▶ The application is installed in a cluster. Each MDB listener port that is defined has its own TCF, which is defined at the server scope. Each TCF does not need clone support checked, but does need a Client ID specified. The client ID field must be a unique value amongst all TCFs defined to use the same broker.

When all the servers in your cluster are started, the MDBs and their listener ports will start successfully. When a message is published, *all of the MDBs will get the message, so all of the cluster members will receive a copy of the message*. This behavior occurs because the broker sees multiple subscriptions, one for each application server.

It would be possible to configure a topology that is somewhere in between these two behaviors (more than one TCF but not one per application server) but the functionality provided would be complicated to work out.

If you are using durable subscriptions in a clustered environment then it is essential that the correct behavior is created through the configuration of your JMS components.

## **DIRECT or QUEUED transport type for TCF**

There is an option to change the communication type when setting up access to a publish/subscribe broker. For the Embedded JMS server and a subset of the WebSphere Business Integration products, it is possible to set the transport to DIRECT.

- ▶ Component: TCF
- ▶ Applies to specific messaging topologies: Yes
- ▶ Requires specific usage of JMS in application: Yes
- ▶ Location of setting:

**Resources -> WebSphere MQ JMS Provider -> WebSphere MQ Topic Connection Factories -> <Your\_TCF> -> Transport Type field**

**Resources -> WebSphere JMS Provider -> WebSphere MQ Topic Connection Factories -> <Your\_TCF> -> Port field**

The WebSphere MQ Real-Time transport (DIRECT) is a protocol which allows client applications to communicate directly with the broker via TCP/IP instead of via intermediate queues. This direct communication and the small overhead of the TCP/IP flows used by the Real-Time Transport, offers potentially higher levels of performance than communicating via message queues.

The DIRECT transport only supports non-durable subscriptions, with non-transactional and non-persistent messaging.

For the embedded messaging there are no other configuration settings needed to change the transport type other than on the TCF.

If you are using WebSphere Business Integration Event Broker or Message Broker version 5 then there is more setup needed and more configuration options available. This is covered in chapter 19 of the redbook *Migration to WebSphere Business Integration Message Broker V5*, SG24-6995.

**Restriction:** DIRECT cannot be used with MDBs running in the message listener service. This is because the DIRECT method of communication does not support the Application Server Facilities (ASF) used by the message listener service. ASF is used to provide concurrency and transactional support.

It is possible to make the message listener service run in a Non-ASF fashion. It would be possible to use MDBs and the DIRECT communication method with non-durable, non-persistent, non-transactional MDBs and the non-ASF support in the message listener service. Information about the non-ASF support can be found in the WebSphere InfoCenter. There are further restrictions in functionality if the message listener service is used in non-ASF mode, not least that it affects the whole message listener service for one application server, not just one listener port.

### 15.5.3 The listener service and listener ports

“Basic workload patterns” on page 600 and “Component relationships when using MDBs” on page 605 have already discussed the way in which workload arrives via the messaging system and the relationships between each of the settings for the message listener service. This section will now provide details about any specific settings that will optimize the message listener service and MDBs.

#### Message listener port maximum sessions

When using MDBs within an application it is recommended that advantage is taken of the ability to process messages simultaneously. This is done by increasing the maximum number of sessions on any listener ports that are configured. By default, Maximum sessions is set to 1.

- ▶ Component: Message Listener
- ▶ Applies to specific messaging topologies: No - as long as using MDBs
- ▶ Requires specific usage of JMS in application: Yes

- Location of setting:

**Servers -> Application Servers -> <Your\_Application\_Server>>  
Message Listener Service -> Listener Ports -> <Your\_Listener\_Port>>  
Maximum sessions** field

Deciding on the number to increase this value by is going to have to involve a performance testing and tuning exercise as there a number of factors that will affect this value. The testing is necessary as there are many variants, making it difficult to estimate an optimum value for Maximum sessions. Ultimately it comes down to two factors:

- Time to complete onMessage method of MDB

The time that it takes to process a message off the message queue is going to determine how long resources will be in use. When a message is passed to an onMessage method of an MDB there will be one listener port session, one session and one listener service thread in use. All of these resources will be locked whilst that message is being processed. The shorter the service time of the onMessage method the less time the resources are locked and so the sooner they are free to process another message.

The average time it takes for the onMessage to complete on an MDB can be found using the Tivoli Performance Viewer. See “Monitoring JMS performance within Tivoli Performance Viewer” on page 679. To be accurate this average time must be measured when a realistic application workload is being tested in a realistic production environment.

- Peak workload arrival rate

The advantage of having multiple sessions is that up to x messages from the destination may be processed simultaneously, where x is the free number of listener port sessions. The peak workload will affect the value for the number of maximum sessions as it defines the time when the system will be at its busiest.

In a system where the time between each message arriving is more than the time it takes to process one message, adding more message listener sessions is not going to give any performance improvement as there will be no messages waiting on the queue.

However if the messages cannot be processed faster than the arrival rate then there is scope for increasing the Maximum sessions value to prevent a backlog of messages occurring.

**Important:** The total processing time for a message includes not only how long the message takes to be processed once it has been delivered to the MDB but also the time it takes waiting to be processed. It is *only* the time that it spends waiting that can be optimized by changing the maximum number of sessions.

Understanding the peak and average workload arrival rates are essential in configuring the optimal value for maximum sessions for your application running in your environment.

If possible, come up with an estimate of the maximum number of messages that will arrive on the message queue over a given amount of time that makes sense in your application (second, minute, etc.).

As with all performance tuning, the correct balance between efficient usage of system resources and maximizing throughput needs to be reached. If the workload does arrive faster than messages can be processed then increasing the maximum number of sessions on a listener port will only increase performance if there is spare resource available on the physical server to perform the simultaneous processing of messages.

It is always worth monitoring the number of sessions in use on a listener port. If a test shows that setting the Maximum sessions to 10 did not perform any faster than Maximum sessions set to 5, check to make sure that all 10 sessions were actually in use using the Tivoli Performance Viewer. It might be the case that not enough workload is arriving to push the system to use all 10 sessions, in which case look at the frequency that the message sender is capable of placing messages on the queue.

There is, however, going to be a limit to increasing the number of Maximum sessions that is not to do with hardware restrictions. This is because there is only one queue reader thread per queue per JVM. This is the thread that is farming out work to the sessions. So as you increase Maximum sessions, the queue reader thread will be kept more busy.

The messaging system topology and WebSphere Application Server topology will make a difference on how the peak workload arrival rate is measured and how long a message takes to process. The topology used in “Clustered Trade3 application WebSphere MQ and WebSphere Business Integration Event Broker” on page 651 has two application servers pointing at the same message queue for redundancy purposes. This means that the message queue is not solely accessed by one message listener service.

Whilst the time to complete the message and the workload arrival rate have a significant impact on how many sessions should listen for messages, they are not

the only settings, and are in themselves affected by a great number of other factors. The only reliable way to determine the appropriate number of maximum sessions on a listener port is through thorough performance testing and tuning.

**Tip:** Having trouble getting the Maximum sessions value to increase above 1 in tests? Your messages are probably being processed faster than the next one is arriving. One way to simulate the peak workload arriving at the message queue is to stop the message listener port in the Administrative Console, but continue to forward messages to the message queue. This will fill up the queue with messages. Once the number of messages on the queue has reached the desired amount, start the message listener port up again.

One final point to reemphasize, increasing the maximum number of sessions on a listener port requires increasing other values too, check “Component relationships when using MDBs” on page 605 to find out what else needs to be increased.

## Maximum messages

This setting controls the maximum number of messages that the listener can process in one session. This setting will only change behavior if the JMS provider is either the MQ JMS or WebSphere JMS provider.

- ▶ Component: Message Listener
- ▶ Applies to specific messaging topologies: No
- ▶ Requires specific usage of JMS in application: Yes
- ▶ Location of setting:

**Servers -> Application Servers -> <Your\_Application\_Server> ->  
Message Listener Service -> Listener Ports -> <Your\_Listener\_Port> ->  
Maximum messages field**

The MQ JMS based providers handle the starting of the MDBs transaction differently than other JMS providers.

When the JMS provider receives one or more messages it retrieves a listener port session, gets the QueueSession from it and passes the message to that QueueSession. The JMS provider then calls start on the listener port session which passes the MDB reference to the QueueSession, spins off a new message listener thread and calls run on the QueueSession. The JMS provider then delivers the message to the onMessage method of the MDB.

The difference between MQ JMS and other providers is that for MQ JMS and the Embedded JMS provider, the listener service begins the transaction before it calls run. This is because WebSphere MQ needs the transaction in order to



retrieve the message, before passing it to the MDB. Other JMS providers are able to receive the message without the MDBs transaction.

This difference in behavior can be exploited using the Maximum messages setting. As the message listener service is starting the transaction it is possible to pass multiple messages to the QueueSession within the context of one transaction, rather than having to have one transaction per message. The number of messages being the value in Maximum messages. Each session will still process the messages in serial fashion.

Changing the value of Maximum messages will reduce the number of transactions that have to be created to process messages. However, this setting might not have an obvious impact on performance, and it changes the way in which those messages are handled:

- ▶ If one message in the batch fails processing with an exception, the entire batch of messages is put back on the queue for processing.
- ▶ Any resource lock held by any of the interactions for the individual messages are held for the duration of the entire batch.
- ▶ Depending on the amount of processing that messages need, and if XA transactions are being used, setting a value greater than 1 could cause the transaction to time out.

The performance improvement will not be obvious when changing this setting as messages will not necessarily be processed faster. The correct ratio between Maximum messages and Maximum sessions would need to be reached.

For example, if there 10 messages on a queue and the Maximum sessions is set to 5, Maximum messages is set to 1, then 5 messages at a time will be processed (assuming enough physical resources). If Maximum messages is set to 5 and Maximum sessions remains at 5, then 5 messages will go to the first session and 5 messages will go to the second session. This means that only 2 messages will be processed at the same time.

The workload will depict how effective changing the Maximum messages setting will be. In the example, if the number of messages is increased to 50 or more then you should start to see some benefit of the reduced overhead of creating transactions.

This setting is more useful when running batches of messages through WebSphere Application Server, rather than dealing with real time requests.

### **Message listener service thread pool**

Each listener port defined on an application server's message listener service will have a maximum number of sessions that it can process at the same time.

Each session requires one message listener thread to be able to operate. The listener service thread pool must have enough threads available at any one time to allow all the listener ports sessions defined under it to start. This means that if there are four listener ports defined, and between them the maximum sessions settings add up to 20, then the maximum size for the thread pool should be no less than 20.

- ▶ Component: Message Listener
- ▶ Applies to specific messaging topologies: No - as long as using MDBs
- ▶ Requires specific usage of JMS in application: No
- ▶ Location of setting:

**Servers -> Application Servers -> <Your\_Application\_Server> ->  
Message Listener Service -> Thread Pool**

The minimum size and thread inactivity time-out should be used together to balance how much JVM resource and heap is taken up by the inactive threads versus the performance impact of having to start up new threads. Depending on your priorities this could mean setting the minimum size to the same as the maximum size to remove the overhead of creating threads. It could mean setting it to 1 and using a large inactivity time-out so that thread pool does not use system resources until threads are needed, but during a busy time the overhead of the threads being created only impacts the first messages to arrive.

## **Underlying MQ JMS connection pooling**

QueueConnections and QueueSessions are pooled at the Java layer by WebSphere Application Server. At a lower level the MQ JMS classes are also pooling the physical connections to the MQ queue manager. This pooling is done automatically by the MQ JMS classes when accessing the Embedded JMS server or WebSphere MQ server. It is possible to change two of the parameters that control the tidying up of idle connections in this pool should you need to.

There is no association between the MQ JMS connection pool and the QCF or TCF connection or session pool. This lower level connection pool will be used by the MQ JMS classes when needed.

### ***MQJMS.POOLING.TIMEOUT***

By default, the MQ JMS pooling will maintain each established idle connection to the queue manager for five minutes. Change this value to specify a different lifetime for an idle connection. The value should be entered in milliseconds.

### ***MQJMS.POOLING.THRESHOLD***

The MQ JMS connection pooling is also controlled by the number of idle connections in it. If the number of idle/unused connections reaches the default of

10 connections then any other connection becoming idle will be destroyed. This is basically the minimum size for the pool. Changing this value will allow you to control how many established connections can survive for the time-out setting above.

Use these two settings to control how many low level connections are available in the pool and how long they will survive if they continue to not be used.

Follow these steps to use these two values.

1. Go to **Servers -> Application Servers -> <Your\_Application\_Server> -> Message Listener Service -> Custom Properties**
2. Click the **New** button.
3. For name specify either:
  - MQJMS.POOLING.THRESHOLD
  - MQJMS.POOLING.TIMEOUT
4. Enter the value you wish to use, and an optional description. Remember the time-out is in milliseconds.
5. Press **OK** and save your configuration. The settings will come into operation the next time the application server is restarted.

### **Use separate QCF and TCF for MDBs**

Each MDB's listener port is associated to a QCF or TCF. The connection factory objects hold the connection and session pools for accessing the messaging system. When configuring the sizes of these pools there always have to be enough sessions and connections available to allow all parts of the application to access the messaging system at peak load. The more variants there are in working out this peak load the harder it is to get the correct figures for the size of the pools.

The number of connections and sessions that are needed by each listener port on an application server can be calculated. The number of connections and sessions used by the Web and EJB container cannot be calculated in the same way as it is reliant on the type and spread of workload, which can vary considerably. Only a percentage of requests that arrive at the Web or EJB container will need to use JMS, unlike the listener port where 100% of requests (messages) need to use JMS.

Creating a separate QCF or TCF definition that will only be used by MDBs means that:

- ▶ There will always be enough resources available in the connection and session pool to run the listener ports that push work to the MDBs. This is regardless of whether the Web or EJB container are experiencing heavy load.
- ▶ There is one less variable to consider when setting up the connection and session pools for the Web and EJB traffic.
- ▶ When monitoring performance it is obvious which connections and sessions are in use by listener ports. In Tivoli Performance Viewer objects are shown by the name of the QCF or TCF they use.

For each QCF or TCF associated to a listener port:

- ▶ Set the minimum connection pool size to 1 and the maximum connection pool size to 2. The listener port will only ever need one QueueConnection, setting the maximum to 2 is just a precaution.

If the MDBs themselves perform or call code that needs to put a message on a queue using the same QCF or TCF, do not use these figures. Depending on how your application works will determine how many more connections are needed to handle the sending of the message from within the MDB's transaction.

- ▶ Set the maximum session pool size to the same value as Maximum sessions in the Listener port configuration.

Once again, if the MDB or code it calls puts a message on a message queue using the same QCF then more sessions might be needed than this.

### **Failure behavior settings**

There are a number of settings that define how the listener service and listener port operate in the event of a failure. Optimizing these settings will have an impact on performance in the event of a failure. Go to “JMS Listener port failure behavior” on page 636 for information.

## **15.5.4 Setting up the QCF / TCF pools**

When using JMS, the QueueConnection and QueueSession pools are the link between the application and the messaging system. There need to be enough connections and sessions available in each QCF or TCF to satisfy the needs of the peak workload.

- ▶ Provider: WebSphere JMS and WebSphere MQ JMS
- ▶ Component: QCF or TCF

- Location of settings:

**Resources -> WebSphere JMS Provider or WebSphere MQ JMS Provider**  
**-> WebSphere MQ Queue Connection Factories -> <Your\_QCF\_or\_TCF>**  
**-> Connection Pool** (from the Additional Properties pane)

**Resources -> WebSphere JMS Provider or WebSphere MQ JMS Provider**  
**-> WebSphere MQ Queue Connection Factories -> <Your\_QCF\_or\_TCF>**  
**-> Session Pools** (from the Additional Properties pane)

Each QCF and TCF will have a different anticipated workload, so it will need configuring to match that workload. Finding the optimum value for the minimum and maximum connections and sessions in the pools can only be achieved through performance testing and tuning.

However, it is possible to help in finding the correct starting place for maximum pool sizes to use in the performance testing. It is how the QCF or TCF is going to be used that will help in deciding what the size the pools should be. Consider the areas listed below when setting up the connection and session pools:

- What is the intended usage of the QCF or TCF?
  - MDB Listener port
  - XA enabled resources
  - Generic JMS
- Are there going to be different QCF or TCF objects for each of the above?

If so then adjust the size of the pools accordingly (minimum and maximum connections for each pool).
- For generic JMS, what is the relationship between QueueConnections and QueueSessions?

See “Component relationships when using generic JMS” on page 608. Remember that each QueueConnection has its own pool of sessions, but each session pool is globally configured on the QCF or TCF.
- How many Web and EJB threads are there defined in the application server?

The Web and EJB containers maximum thread pool sizes act as the controlling mechanism for how many threads could be trying to use the QCF or TCF. Whenever the number of threads is increased or decreased, consider changing the connection and session pool sizes as well. See “Component relationships when using generic JMS” on page 608 for more information.

### 15.5.5 More information

For more information on configuration settings that affect JMS performance including some WebSphere MQ specific settings look at the WebSphere Application Server 5.1 InfoCenter messaging tuning section:

[http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tprf\\_tunejms.html](http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tprf_tunejms.html)

Also take a look at the two supportpacs available about JMS performance with WebSphere MQ:

[http://www-1.ibm.com/support/docview.wss?rs=203&uid=swg24006854&loc=en\\_US&cs=utf-8&lang=en](http://www-1.ibm.com/support/docview.wss?rs=203&uid=swg24006854&loc=en_US&cs=utf-8&lang=en)  
[http://www-1.ibm.com/support/docview.wss?rs=203&uid=swg24006902&loc=en\\_US&cs=utf-8&lang=en](http://www-1.ibm.com/support/docview.wss?rs=203&uid=swg24006902&loc=en_US&cs=utf-8&lang=en)

## 15.6 Optimizing JMS performance in the code

Best practices in the code can help in reducing complexity, maintenance costs and improve performance. Chapters 6 and 16 in the redbook *EJB 2.0 Development with WebSphere Studio Application Developer*, SG24-6819 contain information on designing and developing using message driven beans.

Also, look at the *WebSphere MQ using Java* manual which can be downloaded from:

<http://www.ibm.com/software/integration/mqfamily/library/manualsa/manuals/crosslatest.html>

Below are listed some other areas where good programming can improve performance.

### 15.6.1 Selectors

A selector acts as a filter when waiting to receive a message. If a selector has been specified on the receive action then only messages that match criteria in that selector will be returned. A selector can be setup on MDBs and also coded in generic JMS when doing a receive.

#### **Generic JMS and point-to-point messages**

The WebSphere MQ queue manager (or Embedded JMS server) does not allow message filtering by content. So to implement selectors in point-to-point, MQ JMS will browse a message, parse it, compare it with the selector and then either retrieve it if there is a match or browse the next message. This is termed

*client-side selection*, and has a performance cost as each message must be retrieved to the client to be browsed.

The queue manager does allow filtering by certain fields with certain formats in the MQ header which accompanies all messages. The only two of those fields which are available to a JMS application are:

- ▶ Message ID
- ▶ Correlation ID

When selecting on these fields, in the particular case of equality testing, the MQ JMS code is able to avoid the browse-parse-get loop and simply do a `get-by-messageId` or `get-by-correlationId`. This optimization is termed *server-side selection* as the messages are no longer retrieved to the client first to attempt to match the selector, instead this is done on the server. This only applies if the selector is a simple one, so for example no use of AND or OR with another field in the selector string. Using the server-side selection will make the matching process run faster.

The server side selection will only work if the values for message ID or correlation ID match the WebSphere MQ/Embedded JMS format.

### ***Message ID***

A message ID is generated by the JMS provider and has to conform to the JMS specification of:

ID:nnnnn

where nnnnn is a provider specific value. For Embedded JMS and WebSphere MQ the message ID is a 48 character hexadecimal value and will look like:

ID:414d51205741535f617070315f6a6d73b9f1cc3f2000071c

As it is generated, writing a selector that uses the message ID field will use server side selection.

### ***Correlation ID***

The correlation ID is not generated by the JMS provider and is left to the programmer to set. To take advantage of server side selection the correlation ID needs to be set to the correct format so that WebSphere MQ will recognize it and bypass the costly browse-parse-get loop. There are two ways to do this:

- ▶ Set the correlation ID to the message ID

The message ID is already in the correct format as it is generated. Using this will save having to construct a correlation ID of the correct format and is the recommended method.

- Manually construct the correlation ID to the correct format

This means making a String that is 48 hexadecimal characters. If you wish to use a String that is readable and then convert that into hexadecimal then that String needs to be 24 characters long.

### ***Example***

One common activity which uses a selector is within one application thread, send a request message and wait for the reply message. The reply message is identifiable on the incoming queue by an identifying field, in this example the correlation ID field. Finding the correct message is then a case of using a selector that will match on that correlation ID.

The code to do this would look something like the code fragment in Example 15-1.

#### *Example 15-1 Example receive using a selector string for correlation ID*

---

```
QueueSession qs = qc.createQueueSession(false, QueueSession.AUTO_ACKNOWLEDGE);
String messageSelector = "JMSCorrelationID = '"+correlid+"'";
QueueReceiver receiver;
receiver = qs.createReceiver(myQueue,messageSelector);
TextMessage message = (TextMessage) receiver.receive();
String s = message.getText();
```

---

This code will only do a server side selection if the field `correlid` is encoded in the correct format. There are two choices, and for this example it needs to manually generate a correlation ID, which will then be placed in the request message to be returned in the reply message.

As the correlation ID field is a programmer defined field then the application flow needs to be:

1. Create message.
2. Create the correlation ID using your user defined mechanism. If it is a String then it needs to be 24 characters long. This might mean having to pad it with zeros or spaces.
3. Place this 24 character string in the correlation ID of the message using the `setCorrelationID` method and send the message.
4. For the receive, that correlation ID needs to be converted into the correct format for the selector to use in the code in Example 15-1. This is the format `ID:hexadecimalnumber`. An example of a method that could do this for you is in Example 15-2 on page 633. Given a String this method will return a String that is the characters `ID:` followed by the original String encoded in hexadecimal.



```
public String toHexString(String id) {
    byte[] bytes = id.getBytes();
    char[] result = new char[3+(2*bytes.length)];
    result[0] = 'I';
    result[1] = 'D';
    result[2] = ':';
    final char[] hexits = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
'A', 'B', 'C', 'D', 'E', 'F'};
    int source = 0;
    int target = 3;
    while(source < bytes.length)
    { int d = bytes[source++];
      result[target++] = hexits[(d & 0xF0)>>4];
      result[target++] = hexits[(d & 0x0F)];
    }
    return new String(result);
}
```

---

5. Meanwhile the request message is received and processed by the back end application. The correlation ID is taken from the request message and placed in the response message.
6. The selector matches the correlation ID using server-side selection in the response message.

## **MDBs and selectors**

The selector for an MDB is set in the EJB deployment descriptor. Unlike the previous discussion on generic JMS and selectors, the selection process is done by the code in the message listener service not in code written by the application developer. The message listener service monitors all the queues it is responsible for, and together with the JMS provider, will do the work of matching message to MDB selector.

Keeping the depth of the queue (number of messages on queue) to a minimum always increases the speed of using message selectors as there are less messages to parse. When using MDBs with message selectors in point-to-point, you should always make sure to have all messages that arrive on a queue dealt with. Any messages that do not match the selection criteria for the MDBs monitoring that queue, will be left sitting on the queue. Over time they could build up and slow down the selection process.

Either make sure that the MDBs that use selectors on a message queue, cover all possibilities for arriving messages, or use one of the configuration options to remove the unselected messages.

Depending on your application it could be possible to specify an expiry for messages on the queue.

An alternative is to disable message retention. The message listener service will check all the selector strings of it's MDBs against each message that arrives. If it does not find a match then the default is for this message to be returned to the queue. The setting that is controlling this behavior is MSGRETENTION(YES) being set on the QCF object.

This setting is configurable from the WebSphere Application Server Administrative Console for a QCF under the WebSphere MQ JMS provider. Figure 15-11 shows where this can be found on the QCF page.

		queue manager.
Message Retention	<input checked="" type="checkbox"/> Enable message retention	<i>[i]</i> Select this tick box to specify that unwanted messages are to be left on the queue. Otherwise, unwanted messages are dealt with according to their disposition options.
XA Enabled	<input checked="" type="checkbox"/> Enable XA	<i>[i]</i> Attribute to indicate whether or not the JMS provider is XA enabled or not. This attribute only applies to specialized models of JMSConnectionFactory. It is meaningless for GenericJMSConnectionFactories, as they define such feature enablements through name/value property pairs.
<div>Apply   OK   Reset   Cancel</div>		

Figure 15-11 Disable message retention on the QCF

To disable message retention, uncheck the box next to **Enable message retention**, save the configuration and restart any application server using this QCF. This will now mean that if the message listener service cannot match a message against one of the MDB selectors that are registered on it, the message will be placed on the queue manager's dead letter queue with an appropriate reason code.

**Important:** This setting will change behavior for all parts of the application that are accessing a queue on a queue manager through this QCF. Use this setting *only* when you are confident that it will not affect any other part of the application. Also, each queue can only have one process that is using selectors on it through this QCF. If there is more than one process, for example the message listener service and a QueueReceiver running in a Web container thread, then there is a danger that messages will be lost.

Multiple MDBs with selectors can be pointing at the same QCF (through their listener port) as they are handled by one process, the message listener service.

## 15.6.2 Application defined persistence

Using persistence on a message queue should only be used when needed. See “Persistent Messages” on page 614 to understand more about how this is affected at the JMS code level.

## 15.6.3 Tidying up

When using JDBC to access a database it is the best practice to make sure that all objects are closed once finished with, including if an exception occurs. The same applies when using JMS.

Using some of the JMS objects can involve using objects from a pool as well as taking up JVM resource. The quicker the object is closed, the quicker it can be tidied up and resources re-used. For this reason it is recommended that any object with a `close` method should have `close` run on it as soon as it is no longer needed. Do not rely on just closing top level objects, like sessions or connections.

It is also important that the `close` method of the JMS objects occurs even if there has been an exception thrown. Figure 15-3 shows an example of how the `finally` block should be coded.

*Example 15-3 Sample for finally block for a piece of code that sends a message*

```
finally {
    // Close QueueSender
    try {
        if (sender != null) sender.close();
    } catch (Exception e) {
    }
    // Close QueueSession
    try {
```

```
        if (qs != null) qs.close();
    } catch (Exception e) {
    }
    // Close QueueConnection
    try {
        if (qc != null) qc.close();
    } catch (Exception e) {
    }
}
```

---

## 15.7 JMS Listener port failure behavior

Within WebSphere Application Server, the listener port is the component on the message listener service that links a message driven bean to a QCF and queue destination, and therefore to the underlying messaging system.

There are two scenarios that need to be catered for when configuring the listener port and message listener service.

### 15.7.1 Failure in the listener port

If a listener port should fail to start, or lose its connections to the queue manager then it is possible to make it retry. There are two settings at the message listener service that govern how many times (re)starting a listener port should be attempted and how long the interval should be in between.

These two settings are:

- ▶ **MAX.RECOVERY.RETRIES**

Use this to specify the number of attempts to start the listener port. The default for this setting is 5 attempts.

- ▶ **RECOVERY.RETRY.INTERVAL**

This setting sets the time interval between the retries. The default for this value is 60 seconds.

If these settings are not changed then the listener port will attempt to start once every 60 seconds over 5 minutes. If the listener port has been unable to start in this time then it will not be started, unless done manually.

Change these values to provide an adequate amount of retries so that your application can recover. For example, if you are using hardware or operating system level clustering to provide failover, then these settings need to be configured to keep retrying for the period it typically takes for failover to occur. In

this way the failure of the queue manager or Embedded JMS server will be masked as a blip in throughput rather than the listener port stopping completely.

Follow these steps to use these two values:

1. Go to **Servers -> Application Servers -> <Your\_Application\_Server> -> Message Listener Service -> Custom Properties.**
2. Click the **New** button.
3. For name specify either:
  - MAX.RECOVERY.RETRIES
  - RECOVERY.RETRY.INTERVAL
4. Enter the value you wish to use, and an optional description. Remember the retry interval is in seconds.
5. Press **OK** and save your configuration.

Once completed it will look similar to Figure 15-12.

Application Servers > Trade3Server1 > Message Listener Service >

### Custom Properties

Specifies arbitrary name/value pairs of data, where the name is a property key and the value is a string value which can be used to set internal system configuration properties. [?](#)

Total: 2

☐ Filter

☐ Preferences

<input type="checkbox"/>	Name ^	Value ^	Description ^
<input type="checkbox"/>	<a href="#">MAX.RECOVERY.RETRIES</a>	3	attempts
<input type="checkbox"/>	<a href="#">RECOVERY.RETRY.INTERVAL</a>	300	seconds

Figure 15-12 The listener port retry settings

The settings will come into operation the next time the application server is restarted.

**Attention:** This behavior is slightly different for connections to a message broker. When the JMS provider code attempts a connection to the message broker for publish/subscribe messaging, if there is a problem getting a response from the broker then it will wait for three minutes before timing out. This means that if a listener port is trying to start or restart there will be an additional timeout value added on between retries to the retry interval. This message broker connection timeout is not configurable from within WebSphere Application Server.

## 15.7.2 Failure to process a message

The listener port will retrieve the messages from the message queue and pass them to the MDB. If there is an error in processing a message then the MDB will rollback and the message listener port will return that message to the queue. Of course, once back on the queue this message will be picked up again by the listener port and a never ending cycle could occur.

There are some settings that can stop this occurring. On each listener port it is possible to configure the *maximum retries* setting. By default this setting is set to 0, which means if the processing of one message fails then the listener port will stop. This is good for stopping the message from getting stuck in a never ending loop but it means that all the other messages waiting on the queue will not get picked up.

It is possible to change the maximum retries to a higher number and then this might give the message a chance to succeed. For example, if the message was being used to update a database but access to the database timed out then retrying might eventually work.

Even changing the maximum retries to a higher number might still result in the listener port stopping.

There is a way to stop this behavior occurring and that is through the use of backout queues on WebSphere MQ. Within the configuration of a queue in WebSphere MQ, it is possible to set a backout queue and backout threshold. The backout threshold works in the same way as the listener ports maximum retries. It counts the number of attempts at delivering a message. Upon reaching the backout threshold number of retries, the message is taken from the queue and placed on the queue specified in the backout queue.

These settings for the queue can be set using the GUI as shown in Figure 15-13 on page 639 or using the command line.

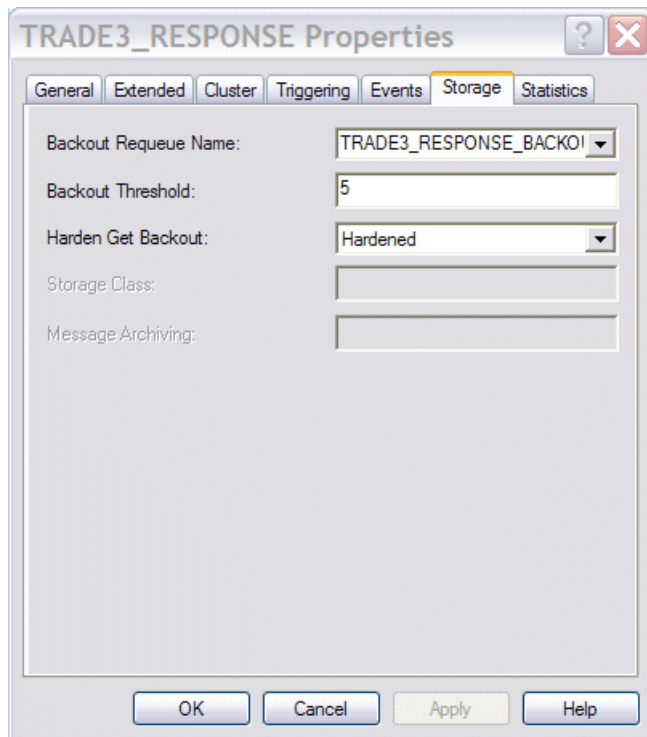


Figure 15-13 The backout queue settings for a message queue.

Make sure to set the Harden Get Backout field to Hardened so that an accurate count of message deliveries is kept.

This now means that the listener port stopping can be avoided by making the backout queue threshold lower than maximum retries. If the maximum retries is set to three on the listener port and the backout threshold set to two, then the message will only ever get delivered twice and so not trigger the setting on the listener port. The message will be placed in the backout queue and perhaps another part of the application can be used to handle the backout queue.

The failed message will be handled and the message listener can continue to run.

## 15.8 Example JMS topologies and scenarios

A lot of the settings that have been discussed have had topology specific benefits. Functionality and configuration are a lot more significant when designing a messaging and application server topology for more than traditional

Web and EJB based applications. With those it is the performance and scalability requirements that often force the choice of topology.

When using JMS, there is not one topology at the top of the food chain that will suit all situations. Using point-to-point, publish/subscribe, MDBs, different messaging products, and combinations of these will all drive a need to have a specific topology to match a specific application and its non-functional requirements.

The choice of topology will depend on the correct balance of non-functional requirements of performance, security, scalability, maintenance, reliability, availability and cost, as well as providing a topology that can give the required function.

The example application that has been used throughout this book is Trade3. Trade3 has specific requirements for its use of JMS. Trade3 uses JMS as an asynchronous communication mechanism between two parts of the application. This means that for Trade3 there is no need to communicate with a back-end system and so a lot of the more complicated topologies do not suit it.

However, this chapter is supposed to be helping you understand JMS, so for the purposes of this section, two scenarios will be described using Trade3 with slightly different requirements for functionality and other criteria. The front end sections of the topology (Web traffic) are not discussed but it is possible to overlay the sample topology in Chapter 7, “Implementing the sample topology” on page 255.

This section is intended to help you use some of the configuration options described in this chapter in working examples. It should also help you begin to understand the thought process in deciding on the best topology for your application. It will not describe all possible topologies, and there are many other topologies. To find out more on choosing a topology take a look at the document *JMS Topologies and Configurations with WebSphere Application Server and WebSphere Studio Version 5* available at:

[http://www.ibm.com/developerworks/websphere/library/techarticles/0310\\_barci\\_a/barcia.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0310_barci_a/barcia.html)

If high availability is important to you then you may also wish to look at the chapters that cover high availability in this redbook. Especially Chapter 11, “WebSphere Embedded JMS server and WebSphere MQ high availability” on page 417.



## 15.8.1 What does Trade3 use JMS for?

Before continuing to the examples it is important that the functional requirements for JMS in Trade3 are clear.

Trade3 integrates both queue based and publish/subscribe MDBs.

### Point-to-point

The queue based TradeBrokerMDB asynchronously processes stock purchase and sell requests. In the Trade3 configuration, if the Order Process mode is set to Asynchronous\_1-Phase or Asynchronous\_2-Phase then instead of an order for some stock being processed immediately, it is handled asynchronously. Upon buying some stock an order is opened in the Trade database and a message sent to a queue. The application then immediately returns to the client. In the background the message is picked up by the TradeBrokerMDB and processed, updating the records in the database to set the order to completed. The next time the client returns to the Web site, they will receive a notification on their Web page that the order completed successfully.

This is using JMS for asynchronous communication within the application. Some important points to note on the influence the functionality has on the topology are:

- ▶ The topology required for this part of Trade3 does not require communication with any back-end systems.
- ▶ Any cluster member can process the order messages as they update the database. This means that if the client is on Trade3Server1 and the message is processed by Trade3Server2, the client will still receive the notification. This is asynchronous communication with no need for affinity.
- ▶ Having one or more JMS servers or WebSphere MQ queue managers will not affect the functionality of the application, nor will using messaging clients.

### Publish/Subscribe

The pub/sub based TradeStreamerMDB subscribes to a message topic called TradeStreamerTopic. Quote data for individual stocks is published to this topic as prices change, providing a real-time streaming quote feature in Trade3. The prices change every time some stock is bought or sold.

Some important points to note on the influence the functionality has on the topology are:

- ▶ If the Trade3 application is clustered, then each application server needs to get the publications on price changes. If this does not occur then the prices will not be kept in synchronization across the cluster. This means having only one message broker doing the work (or one cluster of message brokers).

- ▶ By default, the MDB is using non-durable subscriptions. This avoids any restrictions imposed from using durable subscriptions. See “Client ID and Enable Clone support on a TCF” on page 617 for more information.

Now that we understand how Trade3 needs to use JMS we will move on to apply some of the guidelines described in this chapter and create some topologies.

## 15.8.2 Clustered Trade3 application and the Embedded JMS server

The main purpose of this example topology is to demonstrate how to use multiple Embedded JMS servers in a cell to spread the workload imposed on them by the application servers.

To make this example realistic some basic non-functional requirements have been set.

These requirements need to be specified as it is important to recognize that this example topology is not the *best* topology to use when using embedded JMS, nor is it the best one for Trade3. It is one of a many that could provide what is needed. Each topology benefits certain criteria and needs to be chosen based on those as well as function.

The requirements are:

- ▶ Performance is more important than availability. The application does not have to be available for use continuously. Outages occurring will be tolerated. However, when the system is available it must perform to a high standard.
- ▶ Hardware restriction. The cost of the hardware for this application needs to be kept to a minimum.
- ▶ No licenses of WebSphere MQ will be purchased. This decision has been driven by the fact that Trade3 only uses messaging for asynchronous communication within the WebSphere Application Server cluster. Also there is a reduced need for high availability and management of the messaging system.

There would normally be many other influences in choosing the topology, like workload patterns, but this is just to give some real world justification for this example.

## The chosen embedded JMS topology

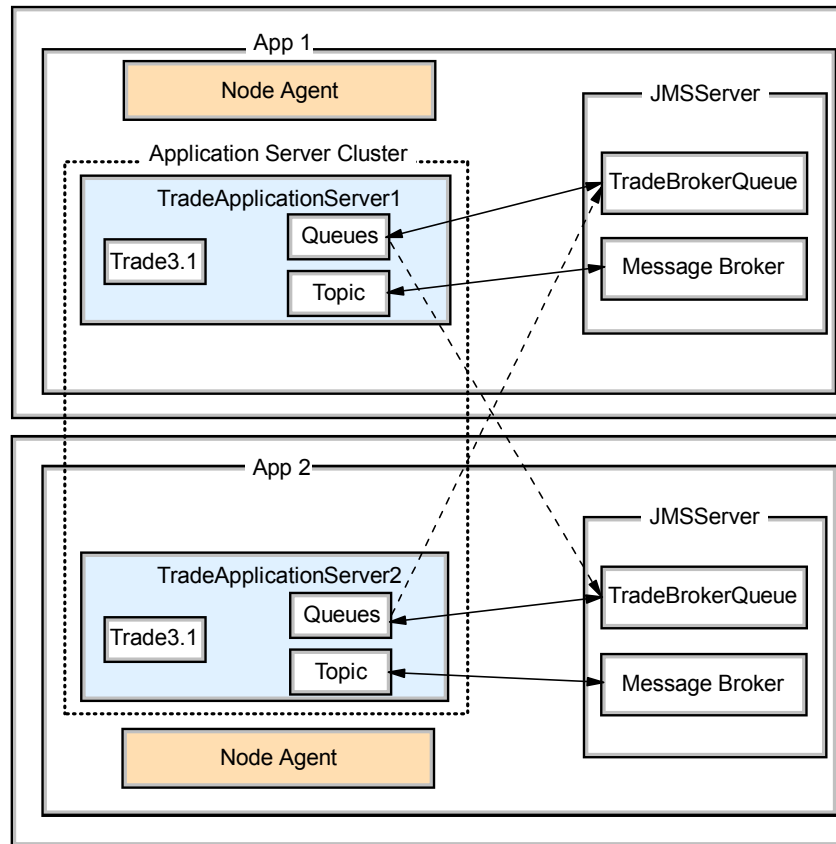


Figure 15-14 Example topology for Trade3 and embedded JMS

This topology has been chosen because:

- ▶ It is low cost. Only two physical machines are needed and there is no requirement for HA hardware or software.
- ▶ Messaging workload will be spread across two JMS servers.
- ▶ The chance of a failure affecting processing of messages is reduced. There is no obvious single point of failure where there would be if just one JMS server was used. But there is still an issue, should either the JMS server or the application server fail on either machine then problems occur. Either messages will be waiting to be processed because the application server has stopped, or the application server will not be able to send messages because the JMS server has stopped.

The impact of the JMS server stopping can be minimized by using persistent messages. When the JMS Server eventually starts up again, the messages it had received will be processed. Failure in sending of messages can be avoided by changing the Trade3 code to have two QCFs to try, one pointing to each JMS server. Upon sending a message if the first QCF fails to respond then the second could be used. This is shown by the dotted line in Figure 15-14 on page 643. (This code change will not be made for this example).

Although the JMS servers are local to the application servers, the Embedded JMS server is always accessed using the CLIENT method. This is because this is the only way it can achieve the correct level of authentication. The performance improvement of having the JMS servers local to the application servers is not as significant as having local MQ servers and using the BINDINGS communication mechanism.

**Note:** Unfortunately using this topology will break one of the functional requirements of the publish/subscribe components of Trade3, only one message broker should be used as all publications need to reach all servers. But it is necessary to do this to show the setup steps and it also serves to demonstrate how easily the topology choice can break the functional requirements for messaging.

## Setting up the topology

This topology is fairly straight forward to configure because the Trade3 example comes with scripts that can do all the work for you. This setup will not be the same as what is shown in Chapter 7, “Implementing the sample topology” on page 255.

It involves:

- ▶ Setting up the WebSphere Application Server cluster
- ▶ Configuring all the JDBC and JMS resources
- ▶ Installing the application
- ▶ Testing the application

It is assumed that IBM WebSphere Application Server Base V5.1 has been installed on to node app1 and app2 with the Embedded JMS server selected on the install options. A copy of IBM WebSphere Application Server Network Deployment V5.1 is also needed. This should have also been installed on app1.

Also, don't forget about the database setup, assuming the database is remote then the IBM DB2 UDB Client version 8.1 will also need to be installed and configured.

**Note:** When installing WebSphere Application Server use a user name that is 12 characters long or less. If a user name of 13 or more characters (like Administrator) is used then the JMS server will be unable to start and you will receive the broker error **BrokerCommandFailedException: Broker command failed: 3008** exception during server startup.

When this is all completed, the nodes app1 and app2 should be federated into the cell. Once this has been done you are ready to carry on.

### ***Setup Trade3 cluster***

With the Trade3.1 application comes a set of scripts to automate the install of Trade3 into a cluster. We are going to use these to speed up the deployment of the example topology.

In Figure 15-14 on page 643 there are two application servers acting as cluster members inside a cluster. These servers need to be configured first so that the listener ports can be configured in a later step.

The cluster can be setup using the `clusterConfig.jacl` script. Its syntax is:

```
wsadmin -f clusterConfig.jacl <clusterName> <nodeName> <serverName>
<weight> <preferLocal>
```

This should all be relatively self explanatory. As there are two servers in this cluster, this will need to be run twice. Here are the commands that we used for this example. These commands are run from the Deployment Manager's bin directory.

```
wsadmin -f t3install\clusterConfig.jacl Trade3Cluster app1 Trade3Server1
2 true
wsadmin -f t3install\clusterConfig.jacl Trade3Cluster app2 Trade3Server2
2 true
```

Checking in the Administrative Console should confirm that two servers and one cluster were created. At this stage you might want to enable the Performance Monitoring Service on both servers for later tests. This can be done in the Administrative Console. See 16.2, "Performance Monitoring Infrastructure" on page 687 for instructions.

### ***Configure JDBC resources***

Trade3 needs access to a database to work. There is another script that will setup the necessary JDBC resources called `createTrade3RDBMSResources.jacl`. Its syntax is:

```
wsadmin -f createTrade3RDBMSResources.jacl dbtype <-cell | nodeName>
rdbmsDriverClassPath rdbmsUser rdbmsPassword \[oracledbhost\] \[oraclesid\]
```

In this example there are two nodes. On each node the location of DB2 driver classes is exactly the same, so to save on configuration the JDBC resources can be created at cell level. For this environment it was done using the command:

```
wsadmin -f t3install\createTrade3RDBMSResources.jacl db2 -cell  
c:/progra~1/sql/lib/java/db2java.zip db2admin db2admin
```

To complete the setup of the database access the Trade database needs to be catalogued locally on each of the clients. Refer to 7.7, “Installing and configuring Trade3.1” on page 298 for details on how to do this.

### ***Configure JMS resources***

In Figure 15-14 on page 643 the two nodes have their own JMS servers. When the federation of the nodes into the cells was undertaken, these JMS servers were automatically created. They will currently be in a stopped state.

To enable Trade3 to access each of these JMS servers, the QCF and TCF objects need to be defined at the correct scope. By defining the QCF and TCF objects at the node level, each application server will use its local JMS server. This is because when the listener port or other calling code does a JNDI lookup for `jms/TradeStreamerTCF` it will resolve to the reference generated at node level. This is explained in detail in “Creation of the JMS provider objects at the correct scope” on page 611.

The first step in creating these objects is to run the script called `createTrade3JMSResources.jacl` that comes with Trade3.1. Its syntax is:

```
wsadmin -f createTrade3JMSResources.jacl nodeName [jmsNode] [serverName]  
[-cell | -node]}
```

- ▶ The `nodeName` is the name of the node on which the application server exists.
- ▶ The `jmsNode` is the name of the node on which the JMS server exists.
- ▶ The `serverName` is the name of the application server that needs its listener ports configured.

To set this up in our environment, the following commands were run:

```
wsadmin -f t3install\createTrade3JMSResources.jacl app1 app1 Trade3Server1  
wsadmin -f t3install\createTrade3JMSResources.jacl app2 app2 Trade3Server2
```

Use the Administrative Console to verify what has been created. The script sets up:

- ▶ Listener ports on each application server pointing to the correct connection factories. It also changes the number of maximum sessions.
- ▶ The queue on the Embedded JMS servers. Using the Administrative Console go to each JMS server to confirm this.

- One QCF, one TCF, one queue definition, and one topic destination at each nodes' scope for the WebSphere JMS provider.

The next step to configure JMS usage is to set up dedicated QCFs and TCFs for the listener ports to use. As discussed in “Use separate QCF and TCF for MDBs” on page 627 this allows for specific configuration of a QCF or TCF for listener port usage and makes defining the size of the connection and session pools simpler.

Follow these steps to setup the new QCF and TCF for use by the listener ports on app1. Once completed these steps need to be repeated for app2.

1. Open the Administrative Console and expand **Resources**. Click **WebSphere JMS Provider**.
2. Set the scope to **Node app1** as shown in Figure 15-15 and click **Apply**. (When you come back to this for app2, remember to change the scope to Node app2).

The screenshot shows the 'Configuration' tab in the WebSphere Administrative Console. The scope is set to 'Cell=dmNetwork, Node=app1'. The 'Node' field is highlighted with a red arrow. There are buttons for 'Browse Nodes', 'Browse Servers', and 'Apply'.

Figure 15-15 Set the scope level to app1

3. Click **WebSphere Queue Connection Factories**. On this page there should already be one QCF called TradeBrokerQCF. This is the QCF that will be used from within all other parts of the application other than the listener port. We are going to create a new one that is just going to be used by the listener port on Trade3Server1.
4. Click **New** and enter the parameters as shown in Table 15-3:

Table 15-3 ListenerPortQCF

Field	Value
Name	TradeBrokerListenerPortQCF
JNDI Name	jms/TradeBrokerLPQCF

Field	Value
Node	app1 (This needs to be app2 when repeating the configuration for app2) This setting points the QCF at the correct JMS server. As there can only be one JMS server per node the node name is sufficient to reference it.
Component-managed Authentication Alias	TradeDataSourceAuthData
Container-managed Authentication Alias	TradeDataSourceAuthData
Enable XA	True

Press **OK** when finished.

- Go back into the QCF by clicking **TradeBrokerListenerPortQCF**. Select **Connection Pool** from the Additional Properties.
- Change Min Connections to **1** and Max Connections to **2**. The listener port will only need one connection. Press **OK**.
- Select **Session Pools**, again from the Additional Properties pane.
- Change Min Connections to **1** and Max Connections to **5**. (The word connections is misleading here, it actually refers to sessions). The listener port will need at most five sessions as this is defined by the maximum sessions setting on the listener port. Press **OK** and then **OK** again.
- Go back to the **WebSphere JMS Provider** and select **WebSphere Topic Connection Factories**. We are going to create a new TCF that is going to be used by the listener port.
- Click **New** and enter the parameters from Table 15-4:

Table 15-4 ListenerPortTCF

Field	Value
Name	TradeStreamerListenerPortTCF
JNDI Name	jms/TradeStreamerLPTCF
Node	app1 (this needs to be app2 when repeating this configuration for app2)
Port	QUEUED
Component-managed Authentication Alias	TradeDataSourceAuthData



Container-managed Authentication Alias	TradeDataSourceAuthData
Enable clone support	Uncheck (false). There is no use of durable subscriptions in this example of Trade3.
Enable XA	Uncheck (false). There is no requirement for 2PC on this object.

Press **OK** when finished.

11. Setup the connection and session pools for this resource by repeating steps 5 on page 648 to 8 on page 648 but for the **TCF** this time.
12. The QCF and TCF are now setup. Now you need to change the listener ports to reference them. Go to **Servers -> Application Servers -> Trade3Server1 -> Message Listener Service -> Listener Ports**. (This will be Trade3Server2 for the app2 node setup.)
13. There are two listener ports. Open each of them in turn and change the Connection factory JNDI name to the newly created versions, **jms/TradeBrokerLPQCF** and **jms/TradeStreamerLPTCF** respectively.
14. **Save** the configuration.
15. Repeat these steps for app2.

The final step for the JMS resources is to enable message persistence. By default the install scripts set the queue and topic destination's persistence to NON\_PERSISTENT to increase performance. This needs to be changed for this example. Each node will have a reference under its scope to the queue and topic destinations as this is how the script runs. (Normally this is not necessary, the queue or topic can usually be specified at cell level as there is not normally any node specific information within it).

Within the **WebSphere JMS Provider** for each node:

1. Open **WebSphere Queue Destinations** and select **TradeBrokerQueue**.
2. Change Persistence to **PERSISTENT** and press **OK**.
3. Open **WebSphere Topic Destinations** and select **TradeStreamerTopic**.
4. Change Persistence to **PERSISTENT** and press **OK**.
5. **Save** the configuration.

The JMS resources are now all setup for the application.

### ***Install the application***

Use the Administrative Console to install the Trade3.ear file. Accept all defaults. See 7.7, "Installing and configuring Trade3.1" on page 298 for installation

instructions if needed. The Trade3.ear file is already configured to use the correct JNDI names and listener ports for this example. The fact that we created extra QCFs and TCFs for the listener ports is masked from the application itself through abstraction.

The application is now installed in your cluster and ready to run.

### ***Testing the application***

The JMS servers need to be started before the application servers can be. In the Administrative Console start up the JMS Servers (select **Servers -> JMS Servers -> <jmsserver\_name> -> Start**) and verify they started correctly in the JMS server logs.

**Tip:** It is possible to get the JMS servers (or any application server) to start automatically when the **startNode** command is run. To change this behavior modify the setting **<MyServer> -> Process Definition -> Monitoring Policy -> Node restart state**.

Once the JMS servers have been started, the application server cluster can be started as well.

The Trade3 application will default to using Synchronous as the order process mechanism on each server. For this reason it is worth change the Trade3 configuration on each of the servers. For each server go to

`http://<server_name>:9080/trade/config`

and change the following configuration parameters:

- ▶ Order processing mode = **Asynchronous\_1-Phase**
- ▶ Enable Operational Trace = **True**
- ▶ Enable Full Trace = **True**

Press **Update config** to complete.

Trade3 will now be up and running. Some tests to try out:

- ▶ Run the Web primitives **PingServletToMDBQueue** and **PingServletToMDBTopic**. Check the **SystemOut.log** of the server to verify the message has been delivered.
- ▶ Stop the listener ports on one of the servers in the **Servers -> Application Servers -> Trade3Server1 -> Message Listener Service -> Listener Ports** menu. Log into Trade3 and place some orders. The orders will not be completed as the MDBs are stopped. Start the listener ports back up again and on the next visit you should receive a notification that the orders were completed.

- ▶ Start the servers with the Performance Monitoring Service enabled. Open up Tivoli Performance Viewer. Stop the listener ports on one of the servers. This means messages will build up on the queue as they arrive. Use Trade3 to buy a lot of stock or use the Web primitives to build up the messages on the queue. Setup the necessary monitors in Tivoli Performance Viewer and start the listener ports back up. Watch the effect in Tivoli Performance Viewer.
- ▶ Stop the listener ports on one of the servers. Purchase some stock then stop the JMS server too. Your orders should remain on the message queue as they are persistent. Start the JMS server back up and then the listener ports and see if the orders are completed.

### 15.8.3 Clustered Trade3 application WebSphere MQ and WebSphere Business Integration Event Broker

The main purpose of this example is to show how WebSphere Application Server, WebSphere MQ and WebSphere Business Integration Event Broker can be used to create a scalable infrastructure. This is to be done through the use of both WebSphere Application Server clustering and WebSphere MQ clustering. This section is aimed at WebSphere Application Server users who have had minimal exposure to WebSphere MQ and WebSphere Business Integration Event Broker.

As with the previous example, to make this realistic some requirements have been set:

- ▶ A new functional requirement has been requested. When stock is bought or sold it needs to be processed by a back-end system first. A response message will then be generated by the back-end and the Trade3 database will be updated as normal.

The reason this new requirement has been specified is because WebSphere MQ clustering only workload manages the sending of messages to remote clustered queues. If the back-end processing is not needed then WebSphere MQ clustering does not aid asynchronous communication between two parts of the same application.

This is explained further in “Initial WebSphere MQ configuration” on page 655.

- ▶ Performance, reliability, and availability are of equal importance. The system must perform well, have minimal outages, and should an outage occur it needs to be able to recover.
- ▶ The application must link into an existing messaging system which is based on WebSphere MQ.

Again, as with the previous example there would normally be other factors that influence the choice of topology, but this should be enough to justify the following topology.

### The chosen WebSphere MQ clustering topology

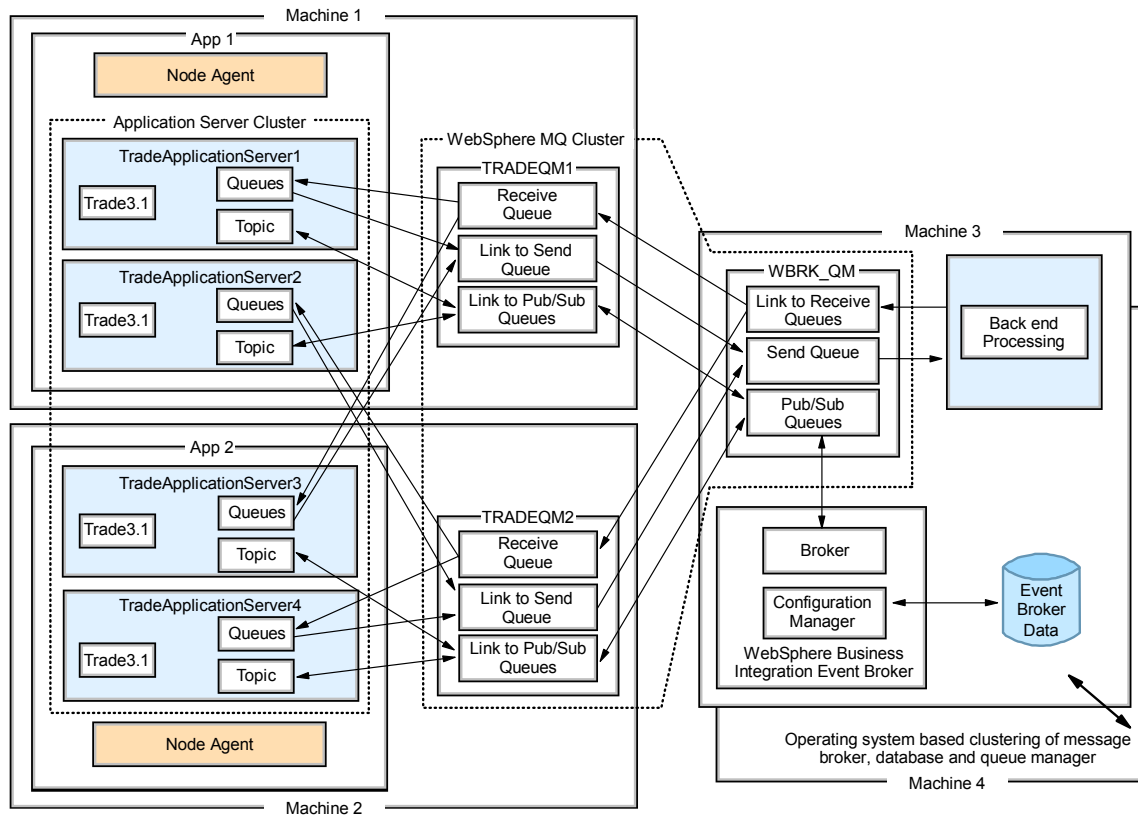


Figure 15-16 Example topology for WebSphere MQ and Event Broker for Trade3 scenario

This topology has been chosen because:

- It provides workload management.

There are four servers in the application server cluster that will have workload distributed to them, both from the Web facing side and from the messaging side. The queue managers also have their workload managed when requests are returned from the back end system. Finally, although not specified on the diagram, there is potential for the broker to be workload managed as well.

- It has been designed for high availability and reliability of WebSphere Application Server.

Four application servers are used instead of two to cover the chance of an application server failing. In Figure 15-14 on page 643 for the previous example, there was an issue that when an application server fails, messages will not be processed on its JMS server. This problem becomes even more important in this topology. In Figure 15-14 on page 643 it is Trade3Server1 that is the only server placing messages on its JMS server. Should Trade3Server1 fail then no more messages will be delivered to app1's JMS server and so the number of messages waiting for processing will depend on what Trade3Server1 was doing when it stopped.

In this topology, messages arriving from the back end are workload managed between any available queue manager. If TRADEQM1 and TRADEQM2 are both up then response messages will continue to arrive at both. If Trade3Server1 was stopped and Trade3Server2 did not exist then messages would continue to arrive at TRADEQM1 but would not get processed.

This is overcome in this example by having one application server on each node pointing at the remote queue manager. Should Trade3Server1 stop, any arriving messages will be picked up by the MDB running on Trade3Server2.

There is still the issue that should TRADEQM1 or TRADEQM2 stop then two of the four application servers will not be able to process order requests that will still be coming in from the front end. This failure in sending of messages can be avoided by changing the Trade3 code to have two QCFs to try, a primary and a backup. Upon sending a message if the first QCF fails to respond then the second could be used. This code change will not be made for this example.

- It has been designed for high availability and reliability of WebSphere MQ.

In this topology, the WebSphere MQ infrastructure uses clustering to provide failover and workload management. This prevents the queue managers becoming a single point of failure for messages arriving from the back end system. See "Initial WebSphere MQ configuration" on page 655.

- It is has been designed for high availability and reliability for WebSphere WebSphere Business Integration Event Broker.

WebSphere Business Integration Event Broker and its supporting software of a WebSphere MQ queue manager and DB2, are made highly available through the use of operating system level clustering software, for example HACMP for AIX. The setup of these products on various types of clustering software is discussed in Chapter 11, "WebSphere Embedded JMS server and WebSphere MQ high availability" on page 417 and thus will not be covered here.

In this topology there is a reliance on the queue managers TRADEQM1 and TRADEQM2 being available for pub/sub messages to be delivered. The

application servers use the two local queue managers as routing mechanisms to reach the message broker.

- ▶ Through use of durable subscriptions, published messages are delivered regardless of the state of the client at publish time.

If the MDB in Trade3 remains using a non-durable subscription as it is currently set, then the subscription to the broker is based on the connection the MDB listener has acquired to its local queue manager. If that connection is broken then the MDB loses its subscription. As this system needs to be reliable and recoverable from failure, a durable subscription needs to be used and the infrastructure setup accordingly. A durable subscription will live past any failure to communicate with the client, storing undeliverable messages until the client comes back online.

- ▶ It provides a topology that fits the functional requirements and allows communication with other WebSphere MQ queue managers outside of the WebSphere Application Server cell.

As with the first example this is not the perfect topology for all applications in all cases.

The back-end application has been included into the additional materials repository of this redbook. Refer to Appendix C, “Additional material” on page 935 for download instructions. It is a very simple EAR file called TradeRedirector.ear, that contains one MDB. This MDB listens for arriving messages on one queue, picks up the messages and puts them back on the response queue. Basic instructions for installing it are included in the EAR file package.

This topology involves more products than in the first example. The following steps for WebSphere MQ and WebSphere Business Integration Event Broker have been designed to be the easiest method of getting a WebSphere Application Server skilled person up and running. However, they might not be the most efficient method to complete the tasks needed.

## Software

The following software was used to perform this example. The steps in this example are for setting this up on a Microsoft Windows platform. It is assumed that before carrying on with these steps, the following software is installed:

### *Machine 1*

- ▶ IBM WebSphere Application Server Base V5.1 - embedded JMS client only.
- ▶ IBM WebSphere Application Server Network Deployment V5.1.
- ▶ IBM WebSphere MQ 5.3.1 CSD 5.
- ▶ IBM DB2 UDB Client version 8.1 fixpack 4.

### **Machine 2**

- ▶ IBM WebSphere Application Server Base V5.1 - embedded JMS client only.
- ▶ IBM WebSphere MQ 5.3.1 CSD 5.
- ▶ IBM DB2 UDB Client version 8.1 fixpack 4.

### **Machine 3**

- ▶ IBM WebSphere MQ 5.3 CSD 5.
- ▶ IBM DB2 UDB version 8.1fixpack 4.
- ▶ IBM WebSphere Business Integration Event Broker 5.0 fixpack 2.
- ▶ IBM WebSphere Application Server Base V5.1 - embedded JMS client only.  
This is to run the back end application.

When installing DB2, WebSphere MQ and WebSphere Business Integration Event Broker accepting the defaults should be enough for this example to work. Where any of the products require fixpacks it is recommended that the fixpacks be applied before starting up the products.

Machine 4 is not going to be covered.

Also, WebSphere Application Server nodes app1 and app2 should be federated into the cell. Once this has been done you are ready to carry on.

## **Initial WebSphere MQ configuration**

In this example the WebSphere MQ configuration underpins both how WebSphere Application Server and WebSphere Business Integration Event Broker operate. WebSphere MQ clustering facilities will be used for workload management of messages, failover and for ease of system administration.

A cluster is a group of queue managers set up in such a way that the queue managers can communicate directly with one another without the need for complex configuration. As well as the WebSphere MQ manuals there is also some good information on configuring WebSphere MQ clusters in chapters 8 and 12 of the redbook *Patterns: Self-Service Application Solutions Using WebSphere V5.0*, SG24-6591.

Figure 15-17 on page 656 shows the queues that need to be configured for this example to work.

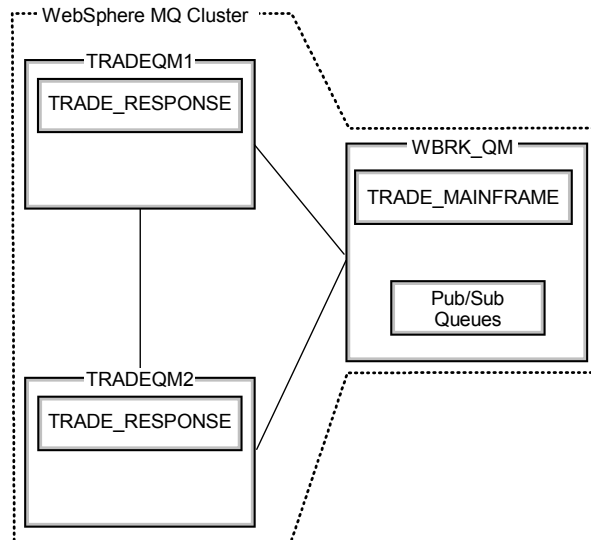


Figure 15-17 The queues in the WebSphere MQ cluster

The way in which MQ clustering works is to first join a number of queue managers together into a cluster (using commands or the GUI). This now means that each queue manager knows about any queue that has been configured as clustered. For example, WBRK\_QM has a queue called TRADE\_MAINFRAME defined under it and set as shared in the cluster. From TRADEQM1's perspective it can now see that there is a queue in the cluster called TRADE\_MAINFRAME and it lives on WBRK\_QM. Any client connecting to TRADEQM1 can simply specify that it wishes to place a message on TRADE\_MAINFRAME and the WebSphere MQ cluster will do the work of getting the message to WBRK\_QM.

This is also how workload management occurs. WebSphere MQ clustering will only workload manage messages when there is more than one remote destination for the message. If a client connects to WBRK\_QM to place a message on TRADE\_RESPONSE then WBRK\_QM can see two TRADE\_RESPONSE queues, one on TRADEQM1 and one on TRADEQM2. The workload management of WebSphere MQ will decide which queue to use. Should one of the queue managers not be available then the message will not be routed there, providing failover too.

However, if a client connects to TRADEQM1 to place a message on TRADE\_RESPONSE then that message will not get workload managed and will go to the local queue on TRADEQM1 even though two queues with the name TRADE\_RESPONSE are visible. (An exception to this is if the client specifies a different queue manager for the location of the queue, but this will still not be workload managed).



Also, once a message is delivered to a queue manager that is the end of its journey until a client picks it up, even if a failure results in that queue manager. In a cluster there is no central repository of messages, a message either exists on the sending queue manager or the receiving queue manager.

Hopefully from this explanation you can now see why it was necessary to include the back-end process. Without the back-end process in this example no workload management or failover of point-to-point messages would occur and so using WebSphere MQ clustering would not be valid for this application. If the only queue managers that existed were TRADEQM1 and TRADEQM2 then there is no need for WebSphere MQ clustering beyond configuration management.

### ***Create queue manager on machine 1***

Follow these steps to setup WebSphere MQ for use in this example:

1. Open the **WebSphere MQ Explorer**.
2. Right-click the **queue managers folder** in the IBM WebSphere MQ Explorer and select **New -> Queue Manager** from the pop-up menu.
3. In Step 1 of the wizard, shown in Figure 15-18 on page 658, set the Queue Manager name to **TRADEQM1** and make sure to specify the Dead Letter Queue of **SYSTEM.DEAD.LETTER.QUEUE**.

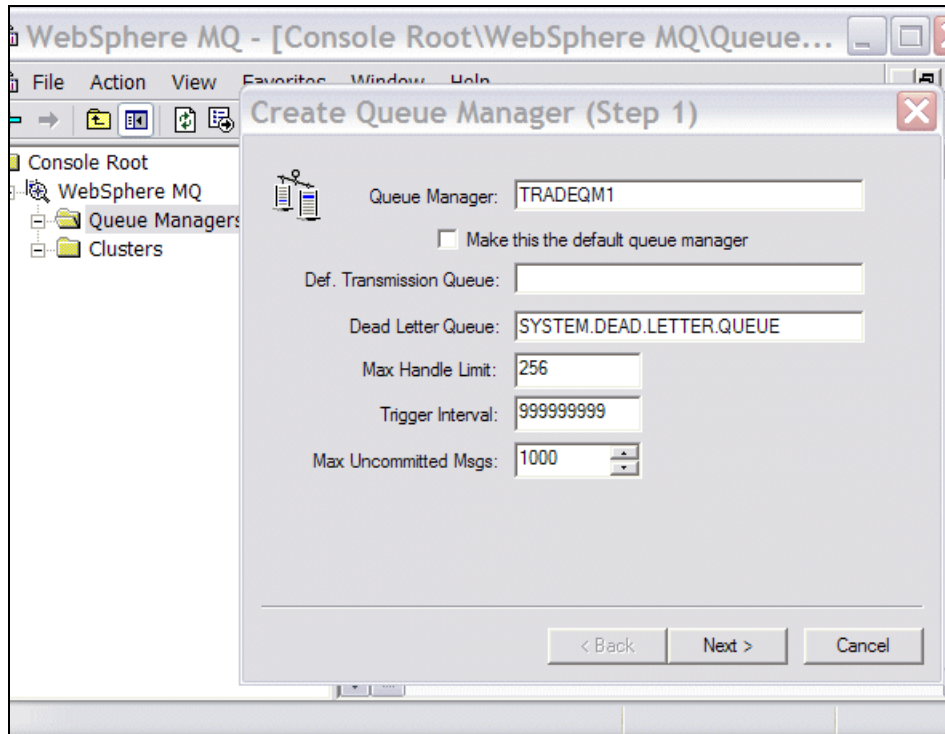


Figure 15-18 Create a new queue manager in WebSphere MQ

4. Click **Next**. Accept the defaults and click **Next** again.
5. On page three of the wizard make sure to check the box next to **Create Server Connection Channel to allow remote administration of the queue manager over TCP/IP**. Click **Next**.
6. On page four of the wizard make sure to enter an unused **Port Number** for the queue managers listener. When creating this example, the Port number 1415 was used. Press **Finish**.
7. The queue manager will be created and started.
8. When using pub/sub and JMS there are a number of extra queues that are needed for it to work. To define these queues first open a **Windows command prompt**.
9. Go to the directory <MQ\_Install\_root>\java\bin. For example to c:\Program Files\IBM\WebSphere MQ\java\bin.
10. Run the command `runmqsc TRADEQM1 < MQJMS_PSQ.mqsc`  
This will create all necessary JMS queues needed for publish/subscribe.

### Create queue manager on machine 2

Repeat the steps from “Create queue manager on machine 1” on page 657 for machine 2, changing the queue manager name to **TRADEQM2** and remembering to choose a **unique port** for the listener.

### Create cluster

To make the steps simpler the creation of the queue manager on machine 3 will be handled later by the setup of WebSphere Business Integration Event Broker. So the next step is to configure the cluster.

1. On machine 1, open the WebSphere MQ Explorer.
2. Right click the **clusters folder** and select **New -> Cluster**.
3. This will bring up a wizard to take you through the steps for configuring the cluster. Click **Next** when you have read the first page.
4. Enter **TRADE3** as the cluster name. Click **Next**.
5. On the third panel select the local queue manager **TRADEQM1** from the options. Click **Next**.
6. For the secondary repository queue manager setup the fields as shown in Figure 15-19. Select **Remote**, enter the queue manager name **TRADEQM2** and the **host name and port** of the queue manager. Click **Next**.

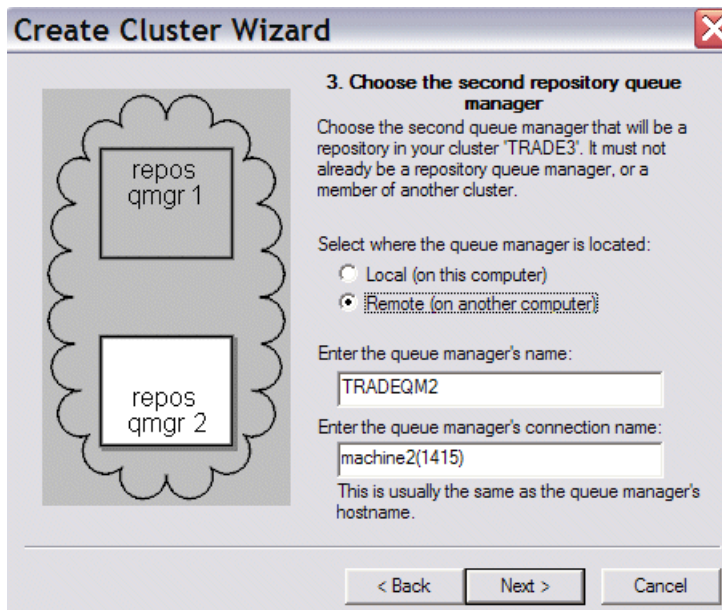


Figure 15-19 Setup second repository queue manager

7. Accept all other defaults in the wizard for channel names, verifying that the host names and ports are valid and click **Finish** to create the cluster.

### **Create TRADE3\_RESPONSE queue on TRADEQM1**

1. Now that the cluster has been created, the queues can be defined. Open the MQ Explorer and expand the **TRADEQM1** folder.
2. Right-click **Queues** and select **New -> Local Queue**.
3. On the general tab of the window that appears enter:
  - Queue name of **TRADE3\_REPSONSE**
  - Default persistence of **Persistent**.
4. Change to the **Cluster** tab. In here select that the queue should be **Shared in cluster** and enter **TRADE3** as the Cluster name. For Default Bind select **Not fixed**.

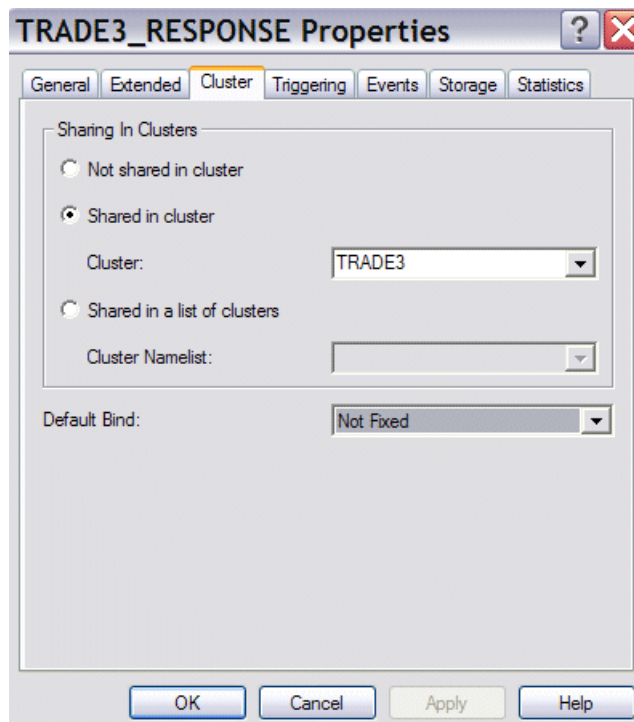


Figure 15-20 Cluster panel of creating a new queue

5. Press **OK** to create the queue.

Under **Clusters -> TRADE3 -> Queue Managers in Cluster -> TRADEQM2 -> Queues** you should now see that a new clustered queue is visible on TRADEQM1.

### **Create TRADE3\_RESPONSE queue on TRADEQM2**

Repeat the steps you just did in “Create TRADE3\_RESPONSE queue on TRADEQM1” on page 660 but define the queue on **TRADEQM2**.

Once this step is complete the main part of the WebSphere MQ configuration is finished.

## **WebSphere Business Integration Event Broker Configuration**

WebSphere Business Integration Event Broker needs DB2 and WebSphere MQ to be installed and running for it to operate. It uses configuration and runtime information stored in DB2 databases, together with queues running in WebSphere MQ, to function as a message broker. The product itself is made of three main components:

- ▶ Configuration manager
- ▶ Message broker
- ▶ Message brokers toolkit

There are several other components but we will use them in this example.

These steps outline what needs to be done to get a simple publish/subscribe broker up and running. This is a very quick run through of the steps needed, offering minimal explanation. For more information refer to the documentation for WebSphere Business Integration Event Broker.

1. Start DB2 on machine 3.
2. When WebSphere Business Integration Event Broker is installed, it defaults to having no licenses available so a broker cannot be started. This count needs to be altered to the number of CPUs that machine 3 has and which are licensed. Open a command prompt and run the command:

– **mqssetcapacity -c 1**

Where, 1 is the number of CPUs.

3. Launch the message broker toolkit by going to **Start -> Programs -> IBM WebSphere Business Integration Event Brokers -> Message Brokers toolkit**.
4. The toolkit is another eclipse based set of tools and might look familiar. When it opens you will be presented with the Welcome screen. (If this does not appear go to the menu option Help -> Welcome....).

5. On the Welcome screen there is the option to **create a default configuration**. Select this option as shown in Figure 15-21.

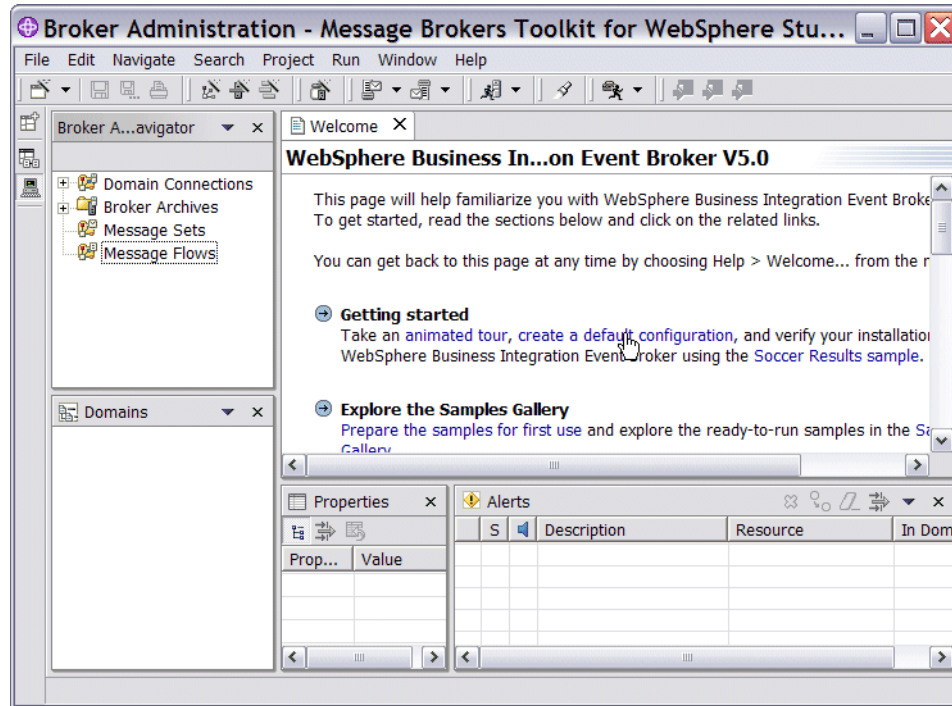


Figure 15-21 The Message brokers toolkit

6. On the first page make sure to use the **user name** and **password** that DB2 is installed and running under as this make installation easier. Click **Next** when done.
7. On the next page, accept the defaults for the queue manager configuration. Make sure to specify a **port number** that is not used. In this example, as with the other queue managers, the port number chosen was 1415. Click **Next**.
8. The wizard will then take you through the names of the broker and configuration managers, and their databases, accept the defaults.
9. On the final page click **Finish** to begin the process of creating all the components and starting them up. This might take a while.  
  
During this process if anything fails then experience has shown that after fixing the problem it is still best to delete the parts that succeeded before re-running the wizard.
10. When everything has been created and started you will receive a success message and you will return to the message broker toolkit.

11. In the broker administrator perspective, in the bottom left view there will now be an active connection shown to the message broker. There will also be an alert message that the default Execution Group is not running. This is shown in Figure 15-22.

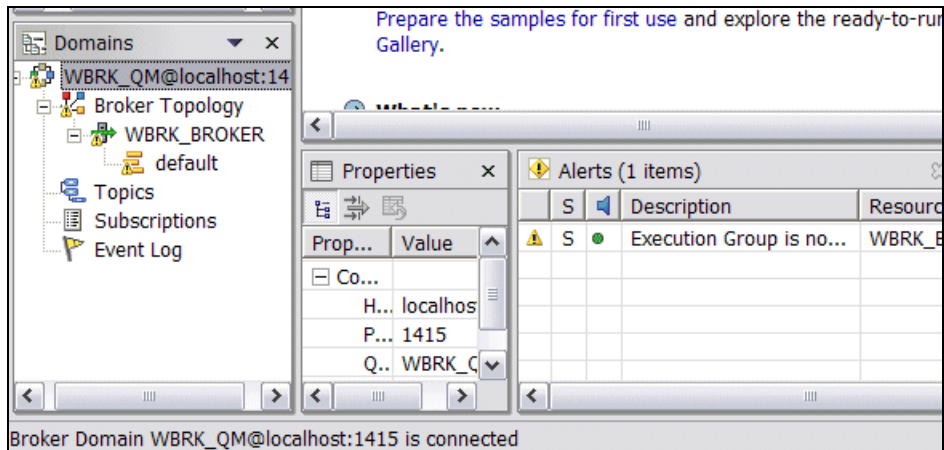


Figure 15-22 Connection to the broker

An execution group is the term used by event broker to group a series of message flows. A message flow is a process that describes the steps a message goes through within the broker, including the point at which it arrives. The default execution group is not running because it does not have any message flows deployed to it. A message flow is needed so that Trade3 can use the broker. There is a sample message flow that will do for the purposes of this example. It will also reduce the number of steps needed to get the broker up and running.

12. Go to the menu option **Help -> Cheat sheets -> Preparing samples for first use**. A new window will open with the message broker toolkit.
13. We have just completed step 1, so go straight to step 2, importing the sample into the workbench. Click the **arrow** to run this step.
14. You will be presented with a new window, select the **Soccer Results Sample** and click **Finish**.
15. Run step 3 in the cheat sheet, creating the runtime resources. Again a window will open, select **Soccer Results Sample** and click **Next**.
16. The next panel will show you the names of the queues that will be created. Select **WBK\_QM** as the queue manager and click **Finish**. You should receive a message that the create sample queues completed successfully.
17. Run the final step to deploy the soccer results message flow. After selecting the **Soccer Results Message Flow** and pressing **Next** you will be presented

with options detailing which broker to publish the message flow to. Accept the defaults and press **Next**, then **Next** again.

18. Finally you will get to a page asking for the execution group. Enter the Execution Group Name of Trade and click **Next**.
19. On the last page of this client window check the box next to **Soccer Messageflows** and click **Finish**. If an error appears saying that the server cannot be started, just click **OK**.
20. Close the cheat sheet window.
21. All that is left to do now is to start the message flow. In the Broker Administrator window, under the Broker Administration Navigator, expand **Message Flows -> Soccer Messageflows -> (default)**. Right click **SoccerPublish.msgflow** and select **Run on Server** as is shown in Figure 15-23.

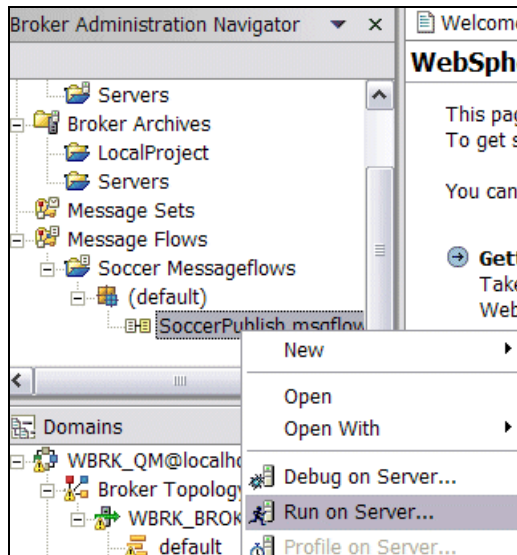


Figure 15-23 Running the soccer sample

22. After pressing **Finish** in the window that appears, you will see that the trade execution group in the domains view will start running. The message broker is now configured and ready for use.



**Important:** At a later stage, for instance after a reboot, you will need to use the commands `mqsisstart configmgr` and `mqsisstart WBRK_BROKER` from the command line to get the broker running. This should be done after DB2 has been started.

Alternatively you can start the respective services.

To complete the setup of machine 3 you will need to install the TradeRedirector.ear file into the application server running on this server and also complete the WebSphere MQ configuration.

### ***Complete WebSphere MQ configuration for WBRK\_QM***

The queue manager WBRK\_QM still needs to be added to the cluster and some queues defined on it. Follow these steps to complete this:

1. Open a Windows command prompt on machine 3.
2. Go to the directory <MQ\_Install\_root>\java\bin. For example:  
c:\Program Files\IBM\WebSphere MQ\java\bin.
3. Run the command `runmqsc WBRK_QM < MQJMS_PSQ.mqsc`  
This will create all of the necessary JMS queues needed for publish/subscribe.
4. Open the WebSphere MQ explorer.
5. Find **WBRK\_QM** and right-click it. Select **All Tasks -> Join a cluster....**
6. In the new window that appears click **Next**.
7. When prompted enter the name of the cluster which is **TRADE3**. Press **Next**.
8. The next page will ask for details of the location of a repository queue manager. Select **Remote** and then enter the queue manager name **TRADEQM1** and the connection name of **machine1 plus the port**, for example machine1(1415). Click **Next**.
9. Upon successful communication with the remote queue manager the prompt will ask for details about the channels to link the queue managers. Accept the defaults and eventually click **Finish**.  
WBRK\_QM has now been added to the cluster. To verify this, take a look at the list of its queues, it should contain references to the TRADE\_RESPONSE queues on the other queue managers.
10. Now that the cluster has been created the last queue can be defined. Open the MQ Explorer and expand the **WBRK\_QM folder**.
11. Right-click **Queues** and select **New -> Local Queue**.

12. On the General tab of the window that appears enter:
  - Queue name of **TRADE\_MAINFRAME**
  - Default persistence of **Persistent**
13. Change to the **Cluster** tab. In here select that the queue should be **Shared in cluster** and enter **TRADE3** as the cluster name. For Default Bind select **Not fixed**.
14. Press **OK** to create the queue.

This completes the setup for WebSphere MQ, there are request and response queues setup for point-to-point. The application servers will communicate with the broker through the WebSphere MQ cluster facilities. When defining the TCFs later, the name of the queue manager that the broker resides on will be used as well as the local queue manager. This allows messages to be automatically routed to the broker's queues.

## WebSphere Application Server configuration

Now that the underlying messaging system has been constructed, the components within WebSphere Application Server can be put together to provide the complete platform for Trade3 to run on.

**Note:** Before setting up any infrastructure, the Trade3 application needs to be altered to use durable instead of non-durable subscriptions. Open up the EAR file using WebSphere Studio Application Developer or the Application Server toolkit and edit the EJB deployment descriptor. On the Beans panel, change the subscription of the TradeStreamerMDB to durable. Save the file then export the new EAR file.

There are four application servers that are going to be deployed. Each of them uses the messaging system in a slightly different way to reduce single points of failure in the topology. This means effective use of the scope setting for each QCF, TCF, queue and topic. Whilst running through these steps keep in mind the overall picture as this is a complicated configuration. Figure 15-24 on page 667 shows what JMS provider objects need to be configured to get this topology to work.

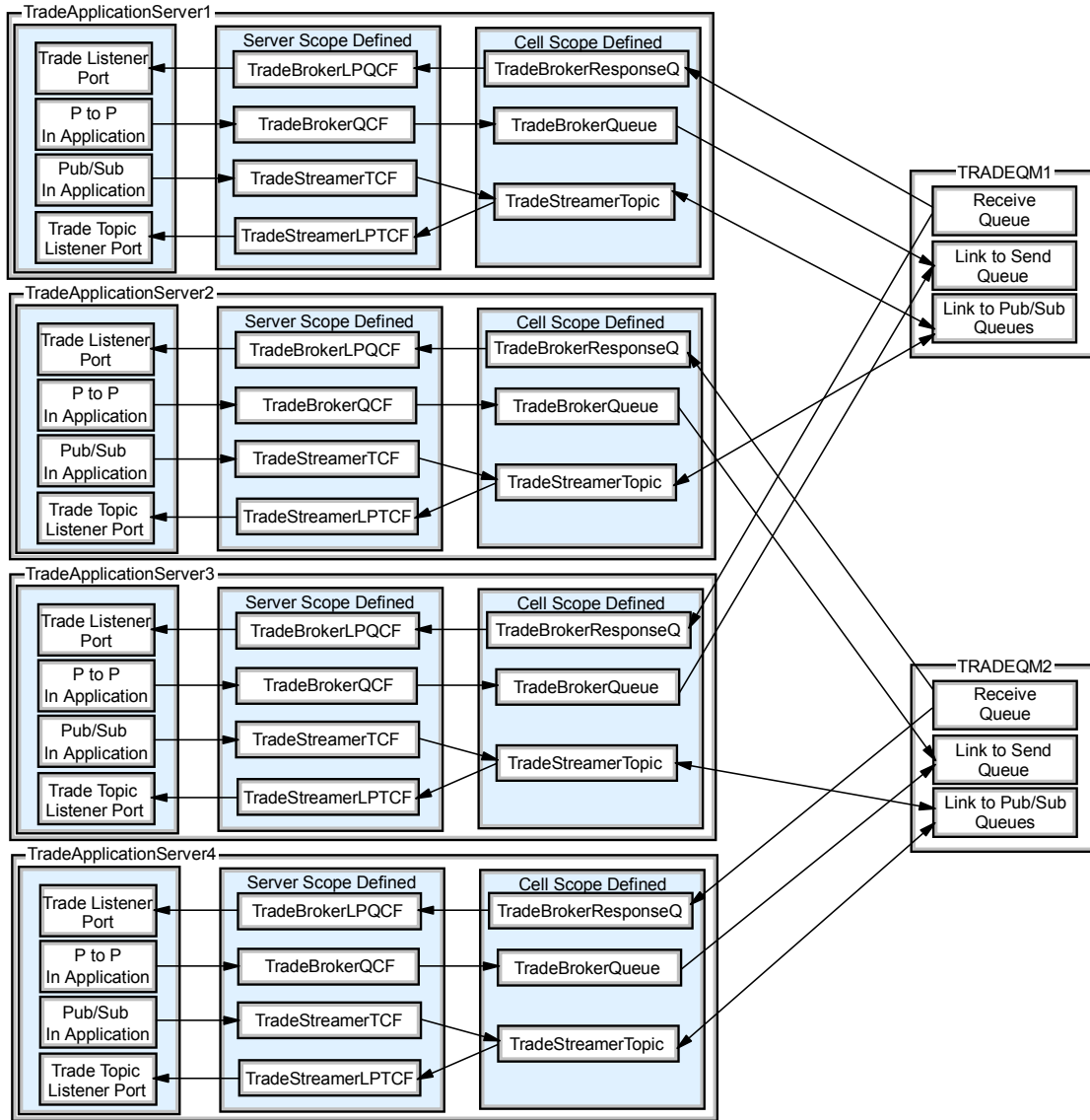


Figure 15-24 Usage of JMS provider objects in sample topology

### Setup Trade3 cluster part 1

In the first example it was possible to use the scripts that come with Trade3.1 to do a lot of the work. Unfortunately the scripts will not assist this time around so the creation of the cluster will be done in two stages, creating the cluster and the first server, then adding new cluster members later on. The advantage of this is that all resources defined at server scope will be copied over to the new servers.

1. Using the WebSphere Administrative Console create a new cluster called **Trade3Cluster**. During the creation of that cluster create one new cluster member called **Trade3Server1** on node **app1**. See 7.5, “Configuring WebSphere clusters” on page 276 if you have questions on how to configure a cluster.
2. When completed save the configuration.

### ***Configure JDBC resources***

Trade3 needs access to a database to work. There is a script that comes with the Trade3.1 package that will setup the necessary JDBC resources, it is called `createTrade3RDBMSResources.jacl`. Its syntax is:

```
wsadmin -f createTrade3RDBMSResources.jacl dbtype <-cell | nodeName>
rdbmsDriverClassPath rdbmsUser rdbmsPassword \[oracledbhost\] \[oraclesid\]
```

In this example there are two nodes. On each node the location of DB2 driver classes is exactly the same, so to save on configuration the JDBC resources can be created at cell level. For this environment it was done using the command:

```
wsadmin -f t3install\createTrade3RDBMSResources.jacl db2 -cell
c:/progra~1/sql1lib/java/db2java.zip db2admin db2admin
```

To complete the setup of the database access, the Trade database needs to be catalogued locally on each of the clients. Refer to 7.7, “Installing and configuring Trade3.1” on page 298 for details on how to do this.

### ***Configure JMS resources part 1***

The first task is to identify how many JMS components need defining within the cell.

#### ► QCFs

By looking at Figure 15-16 on page 652 it is possible to see that for point-to-point messaging each of the application servers need to access the WebSphere MQ cluster in a different way. This is because each possible combination of two queue managers and two forms of communication needs to be covered (CLIENT and BINDINGS). This is achieved by creating all the relevant QCF objects at the server scope, making them visible only to that server.

#### ► TCFs

For publish/subscribe messaging, the MDB has a durable subscription. Each application server needs to receive a copy of any published message so TCFs cannot be shared. Four application servers means there needs to be four durable subscriptions, each made unique by the client ID set on the TCF. (as the MDB name is the same - cloned application). This means defining the

TCF objects at the server scope level as well. More information on this can be found in “Client ID and Enable Clone support on a TCF” on page 617.

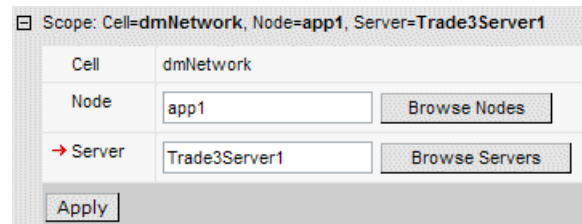
► Queues and topics

Finally the queue destination and topic destination contain no information that is specific to a particular application server, so these can be configured at cell level.

Follow these steps to create the necessary listener ports and JMS provider objects on Trade3Server1.

After completing these steps, Trade3Server1 will be used as a template to create the other servers and the final changes will be made. This will include configuring dedicated QCFs and TCFs for the listener ports to use. As discussed in “Use separate QCF and TCF for MDBs” on page 627 this allows for specific configuration of a QCF or TCF for listener port usage and makes defining the size of the connection and session pools simpler.

1. Open the WebSphere Administrative Console, expand **Resources** and select **WebSphere MQ JMS Provider**.
2. Set the scope to Server **Trade3Server1** (on Node app1) and press **Apply**.



Scope: Cell=dmNetwork, Node=app1, Server=Trade3Server1

Cell	dmNetwork	
Node	app1	Browse Nodes
→ Server	Trade3Server1	Browse Servers
Apply		

Figure 15-25 Set the scope level to app1, Trade3Server1

3. Select **WebSphere MQ Queue Connection Factories**.
4. Create two new QCFs using the information from Table 15-5 and Table 15-6 on page 670. One is for use by the listener port. If a field is not specified then use its default value.

Table 15-5 TradeBrokerQCF

Field	Value
Name	TradeBrokerQCF
JNDI Name	jms/TradeBrokerQCF
Queue Manager	TRADEQM1

Field	Value
Transport Type	BINDINGS
Component-managed Authentication Alias	TradeDataSourceAuthData
Container-managed Authentication Alias	TradeDataSourceAuthData
Enable XA	True

Table 15-6 TradeBrokerListenerPortQCF

Field	Value
Name	TradeBrokerListenerPortQCF
JNDI Name	jms/TradeBrokerLPQCF
Queue Manager	TRADEQM1
Transport Type	BINDINGS
Component-managed Authentication Alias	TradeDataSourceAuthData
Container-managed Authentication Alias	TradeDataSourceAuthData
Enable XA	True

5. Go back into TradeBrokerListenerPortQCF by clicking **TradeBrokerListenerPortQCF**. Select **Connection Pool** from the Additional Properties pane.
6. Change Min Connections to **1** and Max Connections to **2**. The listener port will need only one connection. Press **OK**.
7. Select **Session Pools**, again from the Additional Properties.
8. Change Min Connections to **1** and Max Connections to **5**. The listener port will only need at most 5 sessions as this will be defined by the maximum sessions setting on the listener port. Press **OK** and then **OK** again.
9. Go back to the **WebSphere MQ JMS Provider** and select **WebSphere MQ Topic Connection Factories**.

10. Create two new TCFs using the information in Table 15-7 and Table 15-8 on page 672. One is for use by the listener port. If a field is not specified then use its default value.

Table 15-7 TradeStreamListenerPortTCF

Field	Value
Name	TradeStreamListenerPortTCF
JNDI Name	jms/TradeStreamLPTCF
Transport Type	BINDINGS
Component-managed Authentication Alias	TradeDataSourceAuthData
Container-managed Authentication Alias	TradeDataSourceAuthData
Queue Manager	TRADEQM1
Broker Queue Manager	WBRK_QM
Broker Publication Queue	SOCCER_PUBLICATION (This is the name of the queue that was setup on WBRK_QM. It is being monitored by the broker based on the sample message flow).
Broker Subscription Queue	SOCCER_SUBSCRIPTION
Broker Version	Advanced
Enable clone support	Uncheck (False). Although there is use of durable subscriptions, this TCF is only being used by one application server in the cluster and so there is no issue with MDB listener ports starting up with the same client IDs.
Client ID	Trade3Server1. This unique ID is what allows multiple durable subscriptions on the same topic from within the server cluster. This needs to be different for each server scope.
Enable XA	Uncheck (False). There is no requirement for 2PC on this object.

Table 15-8 TradeStreamerTCF

Field	Value
Name	TradeStreamerTCF
JNDI Name	jms/TradeStreamerTCF
Transport Type	BINDINGS
Component-managed Authentication Alias	TradeDataSourceAuthData
Container-managed Authentication Alias	TradeDataSourceAuthData
Queue Manager	TRADEQM1
Broker Queue Manager	WBRK_QM
Broker Publication Queue	SOCCER_PUBLICATION
Broker Subscription Queue	SOCCER_SUBSCRIPTION
Broker Version	Advanced
Enable clone support	Uncheck (False)
Client ID	Trade3Server1
Enable XA	Uncheck (False). There is no requirement for 2PC on this object.

11. Setup the connection and session pools for this resource by repeating steps 5 on page 670 to 8 on page 670 but for **TradeStreamerListenerPortTCF** this time.
12. Next go back to **WebSphere MQ JMS Provider**. Reset the scope to **cell level**. This is done by removing app1 and Trade3Server1, then pressing **Apply**.
13. Click **WebSphere MQ Queue Destinations**. Two queue destinations need to be created, one to point to the send queue that will go to the back-end process, and one that points to the response queue that will be used by the MDB.



14. Create two new queue destinations using the information from Table 15-9 and Table 15-10. Where no value is given, use the defaults.

Table 15-9 TradeBrokerQueue

Field	Value
Name	TradeBrokerQueue
JNDI Name	jms/TradeBrokerQueue
Persistence	PERSISTENT
Base Queue Name	TRADE_MAINFRAME

Table 15-10 TradeBrokerResponseQueue

Field	Value
Name	TradeBrokerResponseQueue
JNDI Name	jms/TradeBrokerResponseQueue
Persistence	PERSISTENT
Base Queue Name	TRADE3_RESPONSE

15. Go back to the WebSphere MQ JMS Provider and click **WebSphere MQ Topic Destinations**.

16. Create a new topic destination using the information from Table 15-11.

Table 15-11 TradeStreamerTopic

Field	Value
Name	TradeStreamerTopic
JNDI Name	jms/TradeStreamerTopic
Persistence	PERSISTENT
Topic	TradeStreamerTopic

17. Now we need to create the listener ports. Go to **Servers -> Application Servers -> Trade3Server1 -> Message Listener Service -> Listener Ports**.
18. Two listener ports are needed, one for point-to-point and one for publish/subscribe. Create two new listener ports using the information from Table 15-12 on page 674 and Table 15-13 on page 674. Where no value is given for a field use the default.

Table 15-12 tradeport

Field	Value
Name	tradeport
Connection factory JNDI Name	jms/TradeBrokerLPQCF
Destination JNDI Name	jms/TradeBrokerResponseQueue
Maximum Sessions	5
Maximum Retries	10

Table 15-13 tradetopicport

Field	Value
Name	tradetopicport
Connection factory JNDI Name	jms/TradeStreamerLPTCF
Destination JNDI Name	jms/TradeStreamerTopic
Maximum Sessions	1
Maximum Retries	10

19. **Save** the configuration.
20. The install path for IBM WebSphere MQ will need to be set if it is not set already. Go to **Environment -> Manage WebSphere Variables**. Change the scope to **app1** and then set the **MQ\_INSTALL\_ROOT** variable. Do the same for **app2**.
21. Trade3Server1 is now set up and ready to run.
22. At this point you might want to jump straight to installing Trade3 and then running a test. Only Trade3Server1 is configured at this stage but it will be enough to verify that the messaging system is working.

### Setup Trade3 cluster part 2

It is now time to create the other three servers. Trade3Server1 will now be used as a template, reducing the effort to setup all the JMS resources.

1. In the WebSphere Administrative Console go to **Servers -> Clusters -> Trade3Cluster -> Cluster members -> New**.
2. Enter the name **Trade3Server2**, select **app1** as the node. Make sure that **Generate Unique HTTP Ports** is checked, then click **Apply**.
3. Repeat this for servers **Trade3Server3** and **Trade3Server4** which need to be created on **app2**.

4. Once all the servers are in the list press **Next** and then **Finish**.
5. Each of the new application servers will need its Web container transport ports added to the default\_host virtual host before they will accept requests. Go to each application server in turn and write down the ports. Then go to **Environment -> Virtual Hosts -> default\_host -> Host Aliases**. Add the ports if they are not listed here.
6. **Save** the configuration.

### ***Configure JMS resources part 2***

The final step in this setup is to change the QCFs and TCFs for Trade3Server2, 3 and 4 to point to the correct queue manager using the correct transport type. As Trade3Server1 was used as a template for creating these new application servers, all of its resources have come across as well. The listener ports do not need changing. All that needs changing are the QCF and TCF objects.

1. In the scope of **app1/Trade3Server2** on the **WebSphere MQ JMS Provider** change the fields outlined in Table 15-14 to Table 15-17 on page 676 for each QCF or TCF:

*Table 15-14 TradeBrokerQCF*

Field	Value
Name	TradeBrokerQCF
Queue Manager	TRADEQM2
Transport Type	CLIENT

*Table 15-15 TradeBrokerListenerPortQCF*

Field	Value
Name	TradeBrokerListenerPortQCF
Queue Manager	TRADEQM2
Transport Type	CLIENT

*Table 15-16 TradeStreamerTCF*

Field	Value
Name	TradeStreamerTCF
Client ID	Trade3Server2

Table 15-17 TradeStreamerListenPortTCF

Field	Value
Name	TradeStreamerListenerPortTCF
Client ID	Trade3Server2

- In the scope of **app2/Trade3Server3** on the **WebSphere MQ JMS Provider** change the fields shown in Table 15-18 to Table 15-21 for each QCF or TCF:

Table 15-18 TradeBrokerQCF

Field	Value
Name	TradeBrokerQCF
Queue Manager	TRADEQM1
Transport Type	CLIENT

Table 15-19 TradeBrokerListenerPortQCF

Field	Value
Name	TradeBrokerListenerPortQCF
Queue Manager	TRADEQM1
Transport Type	CLIENT

Table 15-20 TradeStreamerTCF

Field	Value
Name	TradeStreamerTCF
Queue Manager	TRADEQM2
Client ID	Trade3Server3

Table 15-21 TradeStreamerListenerPortTCF

Field	Value
Name	TradeStreamerListenerPortTCF
Queue Manager	TRADEQM2
Client ID	Trade3Server3

- In the scope of **app2/Trade3Server4** on the **WebSphere MQ JMS Provider** change the fields outlined in Table 15-22 to Table 15-25 for each QCF or TCF:

Table 15-22 TradeBrokerQCF

Field	Value
Name	TradeBrokerQCF
Queue Manager	TRADEQM2
Transport Type	BINDINGS

Table 15-23 TradeBrokerListenerPortQCF

Field	Value
Name	TradeBrokerListenerPortQCF
Queue Manager	TRADEQM2
Transport Type	BINDINGS

Table 15-24 TradeStreamerTCF

Field	Value
Name	TradeStreamerTCF
Queue Manager	TRADEQM2
Client ID	Trade3Server4

Table 15-25 TradeStreamerListenerPortTCF

Field	Value
Name	TradeStreamerListenerPortTCF
Queue Manager	TRADEQM2
Client ID	Trade3Server4

4. **Save** the configuration. The JMS resources are now all setup for the application.

### ***Install the application***

Use the WebSphere Administrative Console to install the Trade3.ear file. Accept all defaults. The Trade3.ear file is already configured to use the correct JNDI names and listener ports for this example. The fact that we created extra QCFs and TCFs for the listener ports, and where those QCFs and TCFs point, is masked from the application itself.

The application is now installed in your cluster and ready to run.

## ***Testing the application***

The queue managers and message brokers should already be started. If you receive any errors when starting up the application servers verify that the WebSphere MQ cluster is working by looking at the channel and listener status in the WebSphere MQ Explorer. Also, don't forget to start the TradeRedirector application to do the back-end processing.

The Trade3 application will default to using Synchronous as the order process mechanism on each server. For this reason change the Trade3 configuration on each of the servers. For each server go to

`http://<appX>:9080/trade/config`

and change the following configuration parameters:

- ▶ Order processing mode = **Asynchronous\_1-Phase**
- ▶ Enable Operational Trace = **True**
- ▶ Enable Full Trace = **True**

Press **Update config** to complete.

Trade3 will now be up and running. There are many aspects of the failover to try out. Here are a couple of tests to start off with:

- ▶ Run the Web primitives `PingServletToMDBQueue` and `PingServletToMDBTopic`. Check the `SystemOut.log` of the server to verify the message has been delivered.
- ▶ Stop the listener ports on the servers in the **Servers -> Application Servers -> <Trade3ServerX> -> Message Listener Service -> Listener Ports** menu. Log in to Trade3 and place some orders. The orders will not be completed as the MDBs are stopped. Use the WebSphere MQ Explorer to observe workload management delivering the messages to the queues. Start the listener ports back up again and on the next visit you should receive a notification that the orders were completed.
- ▶ Stop the listener ports on one of the servers. Place some orders in Trade3. All the started servers will receive the updated stock price in their logs except the stopped server. Start its message listener service back up and all the publications it missed will be delivered.

## 15.9 Monitoring JMS performance within Tivoli Performance Viewer

This section assumes an understanding of how Tivoli Performance Viewer works. If you need to learn more first then go to Chapter 16, “Server-side performance and analysis tools” on page 685.

The Performance Monitoring service within each application server will provide statistics on the following areas of JMS:

- ▶ MDB executions including average response time of onMessage method
- ▶ Usage of the QCF and TCF connection and session pools
- ▶ Message listener threads

Figure 15-26 shows where these settings can be configured within Tivoli Performance Viewer. This screen shows the available settings when using the Trade3 application and the WebSphere MQ JMS provider.

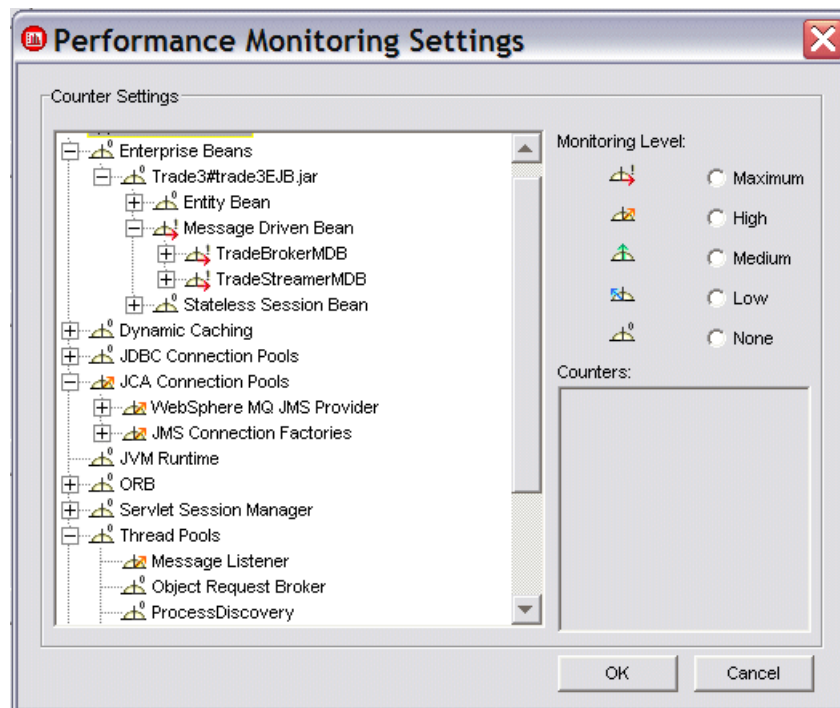


Figure 15-26 Settings for monitoring JMS

By specifying the level of monitoring shown you will be given all the information that Tivoli Performance Viewer can create about JMS.

When using Tivoli Performance Viewer to look at JMS statistics it is not immediately obvious what some of the performance monitors mean. The first issue to overcome is getting all the monitors to appear, then some of the monitors will need explaining.

## Refresh to see all settings

For the QCF, TCF and MDB's methods, the required settings will only appear in the viewer once they have been used at least once. Take a look at the example in Figure 15-27. This is part of a screen that shows the available monitors panel within Tivoli Performance Viewer.

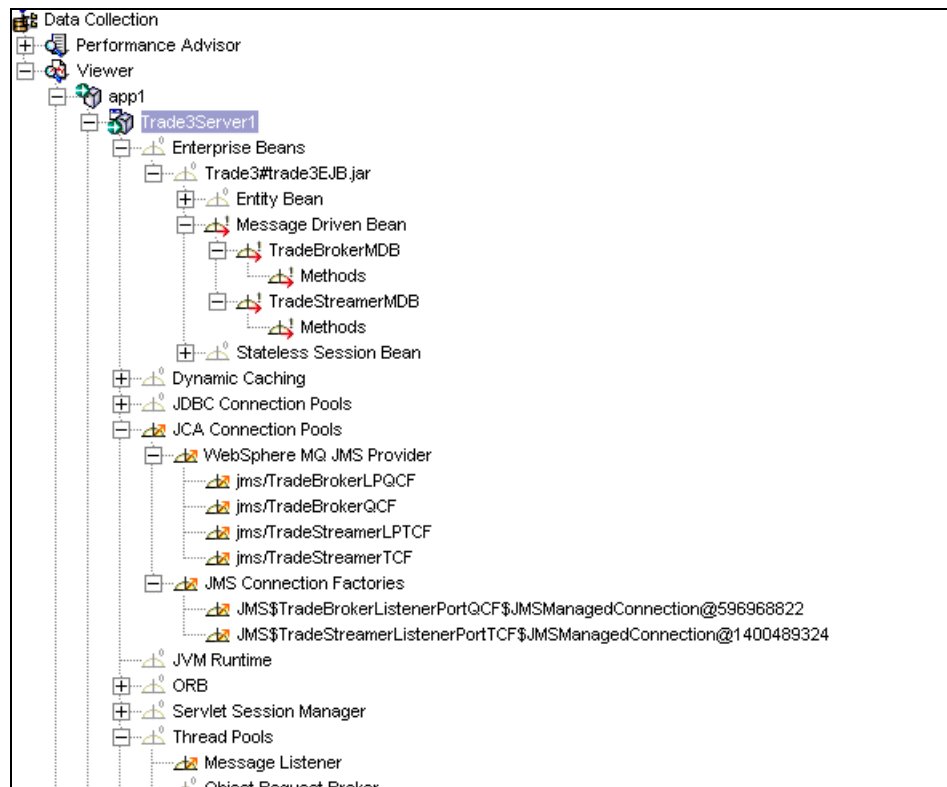


Figure 15-27 The settings prior to refreshing

This is using the Trade3 example again. Upon startup of the application server the monitors were initialized. Tivoli Performance Viewer was opened but some of the information does not appear to be available. For example, the methods under



TradeBrokerMDB are empty. This is because at the stage when the monitors were last updated for this Tivoli Performance Viewer session, no methods had been run on that EJB. To start seeing this information, run the code that will launch the MDB, so in this case use the PingServletToMDBQueue Servlet. Refresh the view.

If you now right-click the methods tag underneath TradeBrokerMDB, the methods will appear, as is shown in Figure 15-28.



Figure 15-28 The view after execution of the code and a refresh of the monitors

## What do the monitors under JCA connection pools mean?

Now that all the necessary monitors are available you will notice that in Figure 15-28 there are multiple entries under the JCA connection pools. This example setup has used the sample topology as described in “Clustered Trade3

application WebSphere MQ and WebSphere Business Integration Event Broker” on page 651. This means two QCFs and two TCFs, one connection factory per listener port and one connection factory for all other JMS requirements.

Each of the monitors listed under WebSphere MQ JMS Provider signifies a QueueConnection pool for the TCF or QCF. As there are four connection factories configured in this example there are four connection pools shown. Within these monitors you will be able to view all the information about the pool of connections, but it does not show how many QueueSessions there are. If you recall, each QueueConnection has its own pool of QueueSession objects.

As a QueueConnection comes into existence it has a session pool created. The monitors under the heading JMS Connection Factories are a list of the QueueConnections in the connection pool.

You may have noticed that in Figure 15-27 on page 680 there are less objects than in Figure 15-28 on page 681. This is because when the application server first starts up, the two listener ports automatically startup as well, and each acquires a QueueConnection. It is possible to identify these QueueConnections in Figure 15-27 on page 680 by their name:

- ▶ JMS\$TradeBrokerListenerPortQCF\$JMSManagedConnection@631227194  
This is the QueueConnection in use from the jms/TradeBrokerLPQCF for the TradeBrokerMDB.
- ▶ JMS\$TradeStreamerListenerPortTCF\$JMSManagedConnection@1122812131  
This is the QueueConnection in use from the jms/TradeStreamerLPTCF for the TradeStreamerMDB.

When workload is run through the application, and the list of JMS Connection Factories is refreshed, more QueueConnections appear, each signifying a different part of the application that has needed a connection. If more than one QueueConnection is needed at the same time, and connection pools settings allow so, then multiple managed connections will appear after a refresh, as has occurred with the TradeStreamerTCF's connections in Figure 15-28 on page 681.

On each of the QueueConnection monitors listed under JMS Connection Factories, it is possible to see how many QueueSessions are in use within that particular session pool. For a connection that is in use by an MDB listener port, the maximum number of QueueSessions on that connection will never go past the maximum sessions setting on the listener port.

Depending on how your connection pools are configured, QueueConnections might be created when needed, and destroyed when idle. This means that this

list under JMS Connection Factories could fluctuate when it is refreshed in the Tivoli Performance Viewer.

Understanding the links between the monitors displayed and need to refresh to get an accurate list of objects will help in performance tuning the product. However, it will present issues if trying to use third party monitoring tools to capture PMI information for test results or for operational alerting.

If you are in this situation then it might be worth configuring the connection pools to have the same minimum and maximum, so that once a connection is created, it does not get destroyed.

**Tip:** The connection pool for a QCF or TCF will not start with the number of connections as specified in minimum connections. It will start at 1 and demand will drive the number of created connections up until the minimum connections is reached. From this point onwards the pool will never drop below the minimum connection pool size.





## Server-side performance and analysis tools

In a production environment, performance and availability of Web applications are critical. In this chapter, we describe how a production environment should be monitored, the types of data available for monitoring, the tools available within WebSphere Application Server to display that data, and tools included to provide guidance on how to optimize performance.

This chapter discusses the following topics:

- ▶ The dimensions of monitoring
- ▶ Performance Monitoring Infrastructure
- ▶ Using Tivoli Performance Viewer
- ▶ Other performance monitoring and management solutions
- ▶ Developing your own monitoring application
- ▶ PMI Request Metrics
- ▶ Dynamic Cache Monitor
- ▶ Monitoring the IBM HTTP Server
- ▶ Log Analyzer
- ▶ ThreadAnalyzer Technology preview
- ▶ Performance Advisors

## 16.1 The dimensions of monitoring

Performance problems can be almost anywhere. The problem can be network and hardware related, backend system related, it can be actual product bugs, or quite often, application design issues.

Understanding the flow used to diagnose a problem helps to establish the monitoring that should be in place for your site to detect and correct performance problems. The first dimension is the "end-user view" - the black box view of your Web site. This is an external perspective of how the overall Web site is performing from an end users view and identifies how long the response time is for an end user. From this black box perspective, it is important to understand the load and response time on your site. To monitor at this level, many industry monitoring tools allow you to inject and monitor synthetic transactions, helping you identify when your Web site experiences a problem.

The second step is to understand the basic health of all the systems and network that make an end user request. This is the "external" view which typically leverages tools and utilities provided with the systems and applications running. In this stage, it is of fundamental importance to understand the health of *every system* involved - including Web servers, application servers, databases, backend systems, etc. If any of the systems has a problem, it may have a rippling effect and cause the "servlet is slow" problem.

This dimension corresponds to the "what resource is constrained" portion of the problem diagnosis. To monitor at this level, WebSphere provides PMI instrumentation and the Tivoli Performance Viewer as a starting point. There are also several industry tools built using PMI instrumentation that provide 24x7 monitoring capabilities.

The third dimension is the application view. This dimension actually understands the application code that is satisfying the end user request. This dimension understands that there are specific servlets that are accessing session beans, to entity CMP beans, to a specific database, etc. This dimension typically comes into play in the in-depth internal understanding of who is using the resource. Typically at this stage, some type of time trace through the application, or thread analysis under load conditions techniques are deployed to isolate areas of the application, and particular interactions with backend systems or databases that are especially slow under load. WebSphere provides the Request Metrics technology as a starting point. In a lot of cases, you start moving into using a lot of the development tools provided such as IBM WebSphere Studio Application Developer.

### 16.1.1 Overview: Collecting and displaying application server data

Table 16-1 shows the types of data that can be collected, the required actions to collect it, and how to view it.

Table 16-1 Performance data collection and viewing

Type of data	Steps/methods of configuration	Viewed with
Performance Monitoring Infrastructure service provides performance data for system resources, WebSphere Application Server, and a customer's application across all transactions. (PMI also includes JVMPI data.)	<ol style="list-style-type: none"><li>1. Set at application server level and Node Agent:<ul style="list-style-type: none"><li>– Administrative Console</li><li>– wsadmin</li></ul></li><li>2. Configure instrumentation level:<ul style="list-style-type: none"><li>– Administrative Console</li><li>– Tivoli Performance Viewer: select <b>File -&gt; Current Activity</b></li><li>– wsadmin</li></ul></li></ol>	<ul style="list-style-type: none"><li>► Tivoli Performance Viewer</li><li>► Monitoring tools using interfaces such as the PMI client, JMX, or Performance Servlet</li></ul>
Request Metrics provides response time data for each individual transaction such as time spent in the Web server, Web container, EJB container, and the backend database.	Set at cell level in the: <ul style="list-style-type: none"><li>– Administrative Console</li><li>– wsadmin</li></ul>	System.out log file monitoring tools using the ARM interface

## 16.2 Performance Monitoring Infrastructure

The second stage of monitoring as described in “The dimensions of monitoring” on page 686 was understanding the basic health of all the systems and network that make up an end user request. For the WebSphere environment, we provide the Performance Monitoring Infrastructure APIs to capture performance data with minimal performance impact to incorporate that data into an overall monitoring solution.

The *Performance Monitoring Infrastructure* (PMI) provides a set of APIs to obtain performance data for system resources, WebSphere Application Server queues, and actual customer application code.

PMI uses a client-server architecture. The server collects performance data in memory within the WebSphere Application Server. This data consists of counters such as servlet response time and data connection pool usage. A client can then

retrieve that data using a Web client, a Java client, or a Java Management Extension (JMX) client. A client is an application that retrieves performance data from one or more servers and processes the data. Clients can include:

- ▶ Graphical user interfaces (GUIs) that display performance data in real time
- ▶ Applications that monitor performance data and trigger different events according to the current values of the data
- ▶ Any other application that needs to receive and process performance data

The PMI components and infrastructure have been extended and updated in WebSphere Application Server V5.0 to support the new management structure and to comply with the Performance Data Framework of the J2EE Management Specification.

PMI is composed of components for collecting performance data on the application server side and components for communicating between runtime components and between the clients and servers. The primary PMI components and related Management Beans (MBeans) are illustrated in Figure 16-1.

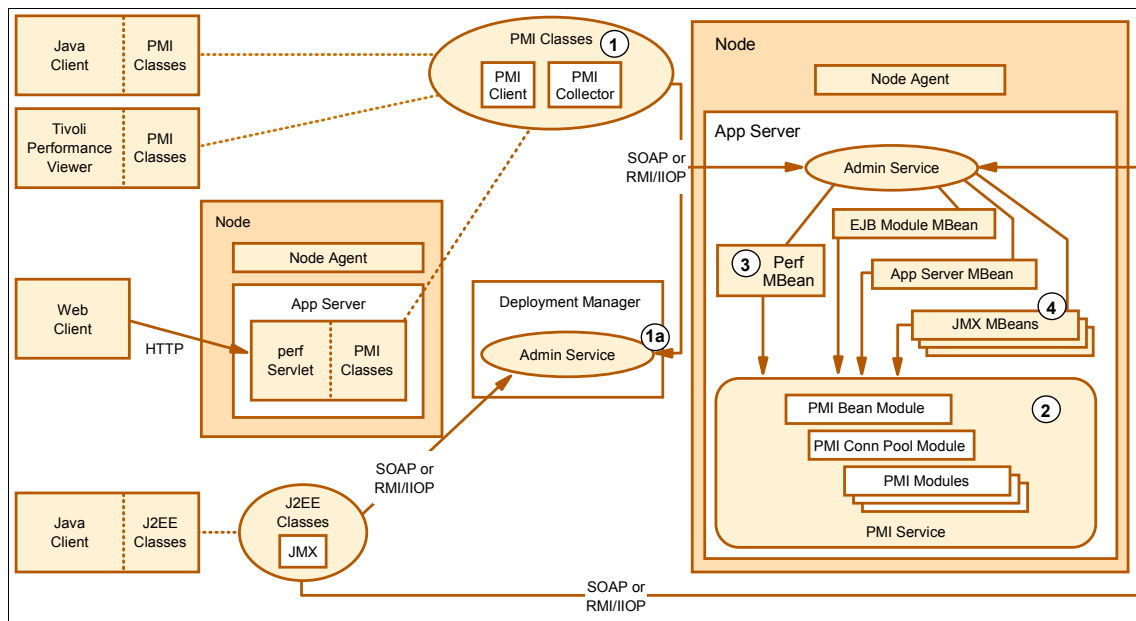


Figure 16-1 Performance monitoring components

1. PMI Client API with its communications implementation, the PMI Collector. The PMI Client API initially contacts the Administrative service of the Deployment Manager (1a) to get a list of nodes, servers and MBeans for the entire cell.



2. PMI service consisting of PMI modules for collecting performance data and methods for instrumenting and retrieving the data from the runtime components. This service contacts the PMI modules upon application server startup, depending on the current configuration.
3. PerfMBean, a JMX management bean used to extract performance data from the PMI modules to the PMI Collector of the PMI Client API.
4. Extensions of the standard JMX MBeans (used for managing components and settings) to support management of performance data. This enables a JMX-based client to retrieve performance data. See Section 19.2, “Java Management Extensions (JMX)” in the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195, for a description of the JMX framework.

The J2EE classes and PMI classes can use either the RMI over IIOP or SOAP protocol to communicate with the Administrative service. In a single-server environment the classes connect to the Administrative service of each individual application server in order to collect performance data. In a Network Deployment environment, the client can choose to connect to the Deployment Manager first to retrieve a list of nodes, servers and MBeans in the cell. Performance data retrieval is subsequently performed in the same way for the two environments.

Each piece of performance data has two components: a static component and a dynamic component. The dynamic component consists of a name and an ID to identify the data, as well as other descriptive attributes that assist the client in processing and displaying the data. The dynamic component consists of information that changes over time, such as the current value of a counter and the time stamp associated with that value.

The PMI client package is part of the PMI application programming interface (API). This can be used to develop your own performance monitoring client. If you are interested in developing your own PMI clients, see “Developing your own monitoring application” on page 723.

### 16.2.1 Performance data classification

PMI provides several different metrics. Each of these metrics is classified into one of the following five types to provide some standardization within the infrastructure. This classification is based on J2EE management specifications and is important to understand when developing a monitoring tool.

- **Count statistic:** Specifies standard count measurements. It consists of a single numeric value and is used to represent data such as counts and sizes. Examples of count statistics include number of times beans were created, number of calls retrieving an object from the pool, and used memory in the JVM runtime.

- ▶ **Boundary statistic:** Specifies standard measurements of the upper and lower limits of the value of an attribute. This classification is currently not being used by PMI.
- ▶ **Range statistic:** Specifies standard measurements of the lowest and highest values an attribute has held as well as its current value. These values can be used for obtaining the number of concurrent invocations to a call method, average number of threads in a pool, or the number of requests that are concurrently processed by the ORB.
- ▶ **Bounded range statistic:** Extends the Range statistic and Boundary statistic interfaces and provides standard measurements of a range that has fixed limits. Examples of use of the Bounded range statistic interface are the number of free connections in the J2C pool, total memory in JVM runtime, and average number of threads in pool.
- ▶ **Time statistic:** Specifies standard timing measurements for a given operation. Examples of time statistics are average response time in milliseconds on the bean methods (home, remote, local) and average connection time in milliseconds until a connection is granted.

**Note:** The performance data classifications have been redefined since the WebSphere Application Server V5 release in order to comply with the J2EE Management Specification. The former classifications (Numeric, Statistical, Load and Group) as well as the new classifications will still be available from the PMI Client API for compatibility reasons.

## 16.2.2 Performance data hierarchy

Performance data is provided in a centralized hierarchy of the following objects to help provide some logical ordering:

- ▶ **Node:** A node represents a physical machine in the WebSphere cell. This is where the Node Agent resides in a Deployment Manager environment.
- ▶ **Server:** A server is a functional unit that provides services to the clients over a network. No performance data is collected for the server itself.
- ▶ **Module:** A module represents a resource category for which performance data is collected. As an example, these are Enterprise JavaBeans, database connection pools, J2C connectors, JVM runtime, Object Request Broker, relational resource adapter, servlet session manager, thread pools, transaction manager, and Web applications.
- ▶ **Submodule:** A submodule represents a fine granularity resource category under the model. Submodules themselves can contain submodules. An example of a submodule is the ORB thread pool, which is a fine granularity resource category under the thread pool module.

- **Counter:** A counter is a data type used to hold performance information for analysis. Examples of counters include the number of active enterprise beans and the time spent responding to a servlet request.

Each resource category (module) has a related set of counters. The counters contain values for performance data that can have an impact on system performance.

Modules can have instances, which are single instantiations of an object of a class. Counters can be assigned to any number of modules and instances. Figure 16-2 shows how this looks like in Tivoli Performance Viewer.

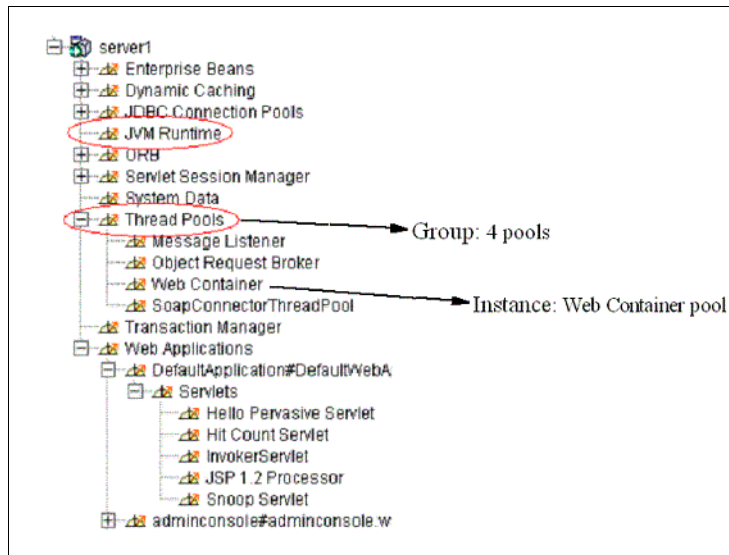


Figure 16-2 Tivoli Performance Viewer

Figure 16-3 on page 692 shows the counter Avg Method RT assigned to both the enterprise beans module and the methods of the Container1.Bean1 instance. Figure 16-3 on page 692 also shows a hierarchy of data collections that are organized for reporting to the Tivoli Performance Viewer.

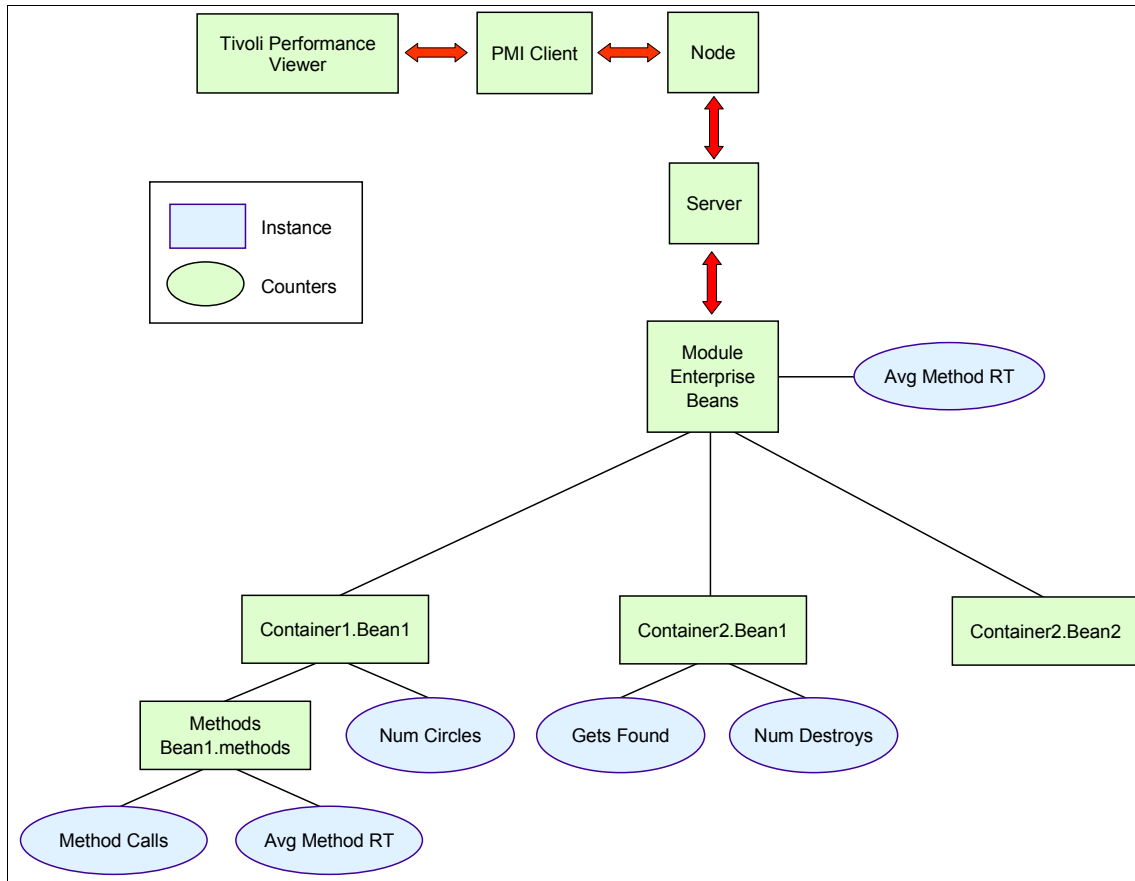


Figure 16-3 Example performance group hierarchy

A subset of counters is available based on the instrumentation level chosen for the particular module or instance. Counters are enabled at the module level and can be enabled or disabled for elements within the module. For example, in the figure, if the Module Enterprise Beans module is enabled, its Avg Method RT counter is enabled by default. However, you can then disable the Avg Method RT counter for the Methods Beans1.methods and the aggregate response time reported for the whole Enterprise Beans module will no longer include Methods Bean1.methods data.

## Performance data organization

PMI data is provided to clients in a hierarchical structure and organized into modules (resource categories) based on runtime components. Each module has a configuration file in XML format that determines its organization. It specifies unique identifiers for each performance data item per module. A client can use

this unique identifier to fetch the performance data's static and dynamic information.

Performance data is collected for each of the following modules (resource categories). Note that the modules denoted with an asterisk (\*) are new since WebSphere Application Server V5:

- ▶ **Enterprise beans**  
Reports load values, response times, and life cycle activities for EJBs. Examples include the average number of active beans and the number of times bean data is loaded or written to the database. It also reports information about the size and the usage of a cache of bean objects (EJB object pool).
- ▶ **JDBC connection pools**  
Reports usage information about connection pools for a database. Examples are the average size of the connection pool, the average number of threads waiting for a connection, the average waiting time in milliseconds for a connection and the average time a connection was in use.
- ▶ **J2C connection pools**  
Reports usage information about the J2EE Connector Architecture that enables EJBs to connect and interact with procedural back-end systems such as CICS® and IMS™. Examples are the number of managed connections (physical connections) and the total number of connections (connection handles).
- ▶ **Java Virtual Machine API (JVM)**  
Reports memory used by a process as reported by the JVM. Examples are the total memory available and the amount of free memory for the JVM. In addition, all performance data that was previously collected in the JVMPI module are now being collected here. Examples are number of garbage collection calls, number of times a thread waits for a lock, and total number of objects allocated in the heap. See “Using JVMPI facility for PMI statics” on page 701 for a description of the JVMPI facility.
- ▶ **Object Request Broker (ORB)\***  
Reports usage information about the Object Request Broker that enables remote clients to instantiate and look up objects in the application server JVM. Examples are lookup time for a object reference before method dispatch, total number of requests sent to the ORB, and the time it takes for a registered portable interceptor to run.

- ▶ **Servlet session manager**  
Reports usage information for HTTP sessions. Examples include the total number of sessions being accessed, the average session life time in milliseconds, and the time taken for writing session data to the persistent store.
- ▶ **Thread pools**  
Reports information about the pool of ORB threads that an application server uses to process remote methods and the Web container pools that are used to process HTTP requests coming into the application server. Examples include the average pool size, the number of threads created and destroyed, and the number of concurrently active threads.
- ▶ **Java Transaction API (JTA)**  
Reports transaction information for the container. Examples include the average number of concurrently active transactions (local and global), the average duration of transactions, and the number of transactions committed, rolled back, and timed out.
- ▶ **Web applications**  
Reports load information for the selected Web application and the installed servlets. Examples are the number of loaded servlets, the number of servlet reloads, the total requests that a servlet has processed, and the response time in milliseconds for servlet requests.
- ▶ **Web services gateway (WSGW)\***  
Reports usage information from the Web services gateway facility. An example of performance data collected is number of synchronous and asynchronous requests and responses.
- ▶ **System data\***  
Reports system level metrics for a node. In WebSphere Application Server Base, this information is available in the application server. In a Network Deployment configuration this information resides in the Node Agent. Examples include CPU utilization and free memory available.

**Important:** In order to have the system data module available on an AIX 4.3.3 node, the bos.perf.perfstat 4.3.3.0 base fileset available on the AIX 4.3.3 installation disk needs to be installed and applied with patches APAR IY24983 and APAR IY27782 available on:

<http://techsupport.services.ibm.com/rs6k/fixdb.html>

AIX 5 ships with these patches already applied.

- ▶ **Workload management (WLM)\***  
Reports information on enterprise bean workload management. Examples include number of WLM clients serviced, server response time, and number of concurrent requests.
- ▶ **Dynamic cache\***  
Reports usage information from the dynamic cache service. Examples include number of client requests, cache misses, and cache hits on disk.
- ▶ **Web services**  
Reports information for Web services. Examples include number of loaded Web services, number of requests delivered and processed, request response time, and average size of requests.

**Restriction:** The following three modules are only available in IBM WebSphere Business Integration.

- ▶ **Alarm Manager\***  
Reports information for the Alarm Manager. Examples include the number of alarms cancelled by the application, number of alarms firing per second, or the number of alarms fired.
- ▶ **Object Pool\***  
Reports information for Object Pools. Examples include the total number of objects created, number of objects requested from the pool, number of objects returned to the pool, and average number of idle object instances in the pool.
- ▶ **Scheduler\***  
Reports information for the Scheduler service. Examples include the number of tasks that failed to execute, the number of tasks executed successfully, number of tasks executed per second, and many more.

### 16.2.3 Performance data counters

Each resource category (module) has its own set of performance data counters. Those counters have particular properties, for example the rating impact and data type.

A complete list of all performance data counters for each resource category is included in the WebSphere Application Server V5.1 InfoCenter article “Performance data organization”. There is a separate table for each resource category. As an example, the ORB counters are shown in Table 16-2 on page 696.

Table 16-2 Counter information for ORB service PMI Client package

Name	Description	Version	Granularity	Type	Level
reference LookupTime	The time (in milliseconds) to look up an object reference before method dispatch can be carried out	5.0	ORB	Time Statistic	Medium
numRequest	The total number of requests sent to the ORB	5.0	ORB	Count Statistic	Low
concurrent Requests	The number of requests that are concurrently processed by the ORB	5.0	ORB	Range Statistic	High
processing Time	The time (in milliseconds) it takes a registered portable interceptor to run	5.0	per interceptor	Time Statistic	Medium

In Table 16-2:

- ▶ Version refers to the version of WebSphere Application Server when the counter was introduced into the PMI framework.
- ▶ Granularity refers to the unit to which data collection is applied for that counter.
- ▶ Type refers to the performance data classification as described in 16.2.1, “Performance data classification” on page 689.
- ▶ Level refers to the instrumentation level for which this counter is included. We will cover instrumentation levels in 16.2.4, “Instrumentation levels” on page 696.

## 16.2.4 Instrumentation levels

The resources in a WebSphere cell are instrumented so that statistical data can be collected. Instrumentation refers to the mechanism by which some aspect of the running system is measured (analogous to a meter attached to a resource). Each resource category has an instrumentation level (different from the impact rating for the counters in that category). The instrumentation level determines which counters are available to be collected for that category and correspond to the level of performance impact a particular grouping of counters will have on the customer environment.

For example, if a resource category has an instrumentation level setting of low, only counters having a low impact rating are available for selection. If the instrumentation level is set to medium, then counters having low and medium impact ratings are available for selection. Similarly, when the instrumentation level is set to high, all counters with low, medium, and high impact ratings are available for selection.



An instrumentation level can also be set to maximum, which enables the availability of all counters and, in addition, increases the level of granularity when reporting on enterprise methods. This setting has a higher impact on a system's performance. Conversely, the instrumentation level can be set to none, which disables performance reporting and eliminates any impact of monitoring on system performance. Initially, the instrumentation levels are set to none.

## 16.2.5 Enabling the PMI service

In order to monitor a resource with Tivoli Performance Viewer or any PMI or JMX client, the PMI service of the application server associated with that resource has to be enabled. The PMI service can be enabled from the Performance Monitoring Service configuration pane in the Administrative Console or by using the **wsadmin** command interface. When running WebSphere Application Server Network Deployment, be sure to enable the PMI service in both the application server and in the Node Agent through the Administrative Console or wsadmin.

### Using Administrative Console to enable the PMI service for the application server

In order to enable the PMI service from the Administrative Console, open the Performance Monitoring Service properties configuration pane by using the following steps:

1. Expand the **Servers** folder from the navigation tree.
2. Click **Application Servers** from the Servers folder.
3. Click the name of your application server (for example server1) from the list of application servers in the workspace.
4. Click the **Performance Monitoring Service** entry in the Additional Properties pane of the workspace. The Performance Monitoring Service properties configuration pane opens in the workspace.
5. To enable the PMI service of this application server, select the **Startup** check box.
6. Save your configuration and restart the application server.

[Application Servers](#) > [server1](#) >

## Performance Monitoring Service

Configuration and Runtime Settings for Performance Monitoring Infrastructure (PMI) ⓘ

**Runtime** **Configuration**

### General Properties

Startup	<input checked="" type="checkbox"/>	ⓘ Specifies whether the server will attempt to start specified service when the server starts.
Initial specification level	<input type="radio"/> None - All modules below set to "N" (None). <input type="radio"/> Standard - All modules below set to "H" (High) <input checked="" type="radio"/> Custom - Modify, add or remove the modules from the below list.	ⓘ A Performance Monitoring Infrastructure (PMI) specification string that stores PMI specification levels for all components in the server. N,L,M,H,X represent None,Low,Medium,High,Medium respectively.

beanModule=N  
cacheModule=N

Figure 16-4 Using Administrative Console to enable the PMI service

## Using Administrative Console to enable the PMI service for the Node Agent

In a Network Deployment environment, you need to enable the PMI service also for the Node Agent(s). Follow these steps:

1. Open the Administrative Console and select **System Administration -> Node Agents** from the navigation tree.
2. Click **nodeagent**.
3. Select **Performance Monitoring Service** from the Additional Properties.
4. Select the **Startup** check box.
5. (Optional) Select the PMI modules and levels to set the initial specification level field.
6. Click **Apply** or **OK**.
7. Save your configuration changes and restart the Node Agent.

The changes take affect after a restart of the Node Agent.

## Using wsadmin to enable the PMI service

In order to configure the PMI service of a specific application server, a reference to the PMI service configuration object of that application server is needed. All PMI service configuration objects can be listed using the **wsadmin list PMIService** command:

```

C:\WebSphere\AppServer\bin>wsadmin
WASX7209I: Connected to process "dmgr" on node net1Manager using SOAP
connector; The type of process is: DeploymentManager
WASX7029I: For help, enter: "$Help help"

wsadmin>$AdminConfig list PMIService

(cells/net1Network/nodes/node1/servers/CSV1:server.xml#PMIService_1)
(cells/net1Network/nodes/node1/servers/CSV2:server.xml#PMIService_1)
(cells/net1Network/nodes/node1/servers/nodeagent:server.xml#PMIService_1)
(cells/net1Network/nodes/node1/servers/server1:server.xml#PMIService_1)
(cells/net1Network/nodes/net1Manager/servers/dmgr:server.xml#PMIService_1)

wsadmin>

```

Figure 16-5 Using **wsadmin** to list the PMI service configuration objects

Each line of output contains the PMIService configuration ID that can be used for referencing the PMIService component of a specific application server.

To enable performance data monitoring, use the following **wsadmin modify** command with your specific PMI service configuration ID:

```

wsadmin> $AdminConfig modify \
(cells/net1Network/nodes/node1/servers/server1: \
server.xml#PMIService_1) {{enable true}}

```

The configuration needs to be saved before restarting the application server. Use the **wsadmin save** command to save the configuration:

```

wsadmin> $AdminConfig save

```

To restart the application server, use these **wsadmin** commands (in a single-server environment, don't specify the node in the **startServer** command):

```

wsadmin> $AdminControl stopServer server1
wsadmin> $AdminControl startServer {server1} {net1}

```

To disable performance data collection, use the following **wsadmin modify** command (save the configuration and restart the application server for the change to take effect):

```

wsadmin> $AdminConfig modify \
(cells/net1Network/nodes/node1/servers/server1: \
server.xml#PMIService_1) {{enable false}}

```

## 16.2.6 Setting instrumentation levels

The instrumentation level can be set by using:

- ▶ WebSphere Administrative Console
- ▶ **wsadmin** command interface
- ▶ Tivoli Performance Viewer

To get you started we discuss setting instrumentation levels using the first two methods.

For information on setting instrumentation levels using the Tivoli Performance Monitor, see 16.3.4, “Setting the instrumentation levels” on page 712.

### Using the Administrative Console to set instrumentation levels

The instrumentation levels are set using the same configuration panel as was used to enable the PMI service. To set the instrumentation levels to be used at application startup as shown in Figure 16-4 on page 698:

1. Expand the **Servers** folder from the navigation tree.
2. Click **Application Servers** from the Servers folder.
3. Click the name of your application server (for example, server1) from the list of application servers in the workspace.
4. Click the **Performance Monitoring Service** entry in the Additional Properties pane of the workspace.
5. Select **None**, **Standard**, or **Custom**. If you choose Custom, type over the level in the window. N,L,M,H,X represent none, low, medium, high, and maximum respectively.
6. Click **OK** and save your changes.

**Tip:** After saving the initial instrumentation level configuration, the Performance Monitoring Service properties pane shows two pages. The Configuration page is used to set the initial settings and the Runtime page is used to alter the instrumentation level at runtime without having to restart the application server.

## Using the wsadmin command to set instrumentation levels

To set the overall instrumentation level to low for the server1 application server, use the **wsadmin set** commands as shown in Example 16-1.

*Example 16-1 Using wsadmin to set instrumentation levels*

---

```
wsadmin> set perfObjRef [$AdminControl makeObjectName \  
    [$AdminControl completeObjectName type=Perf,process=server1,*]]  
wsadmin> set params [java::new {java.lang.Object[]} 2]  
wsadmin> $params set 0 [java::new java.lang.String pmi=L]  
wsadmin> $params set 1 [java::new java.lang.Boolean true]  
wsadmin> set sigs [java::new {java.lang.String[]} 2]  
wsadmin> $sigs set 0 java.lang.String  
wsadmin> $sigs set 1 java.lang.Boolean  
wsadmin> $AdminControl invoke_jmx $perfObjRef setInstrumentationLevel \  
    $params $sigs
```

---

The `setInstrumentationLevel` method call doesn't produce any output. You can use the Tivoli Performance Viewer as a quick way to check that the commands actually had an effect on the instrumentation level setting.

The string passed in as the first parameter to the `setInstrumentationLevel` method (here: `pmi=L`) specifies the instrumentation level setting (possible values are N, L, M, H, X). This string can hold the overall instrumentation level or a compound of instrumentation levels for each module and/or submodule like this:

```
java::new java.lang.String \  
    beanModule=H:connectionPoolModule=H:jvmRuntimeModule=L:orbPerfModule=H
```

### 16.2.7 Using JVMPi facility for PMI statics

The Java Virtual Machine Profiler Interface (JVMPi) is a facility of the JVM used to enable a more comprehensive performance analysis. By enabling the JVMPi interface, the Performance Monitoring Infrastructure can provide more comprehensive performance data such as statistics on garbage collection.

JVMPi is a two-way function call interface between the JVM and an in-process profiler agent. The JVM notifies the profiler agent of various events, such as heap allocations and thread starts. The profiler agent can activate or inactivate specific event notifications, based on the needs of the profiler.

All JVMPi performance data is collected by the JVM module, but JVMPi needs to be enabled for the module to update its counters.

The JVMPi facility is available on the Windows, Unix, and Linux platforms.

## Performance data provided by JVMPI

Below are the statistics that PMI provides through the use of JVMPI:

- ▶ Garbage collector
  - Number of garbage collection calls
  - Average time in milliseconds between garbage collection calls
  - Average duration in milliseconds of a garbage collection call
- ▶ Monitor
  - Number of times that a thread waits for a lock
  - Average time that a thread waits for a lock
- ▶ Object
  - Number of objects allocated
  - Number of objects freed from heap
  - Number of objects moved in heap
- ▶ Thread
  - Number of threads started
  - Number of threads died

## Enabling JVMPI from the Administrative Console

To enable JVMPI reporting for each individual application server or Node Agent, do the following in the WebSphere Administrative Console:

1. Select **Servers -> Application Servers** or **System Administration -> Node Agents** in the console navigation tree (depending on the JVM you would like to profile).
2. Select the application server or Node Agent from the list of application servers or Node Agents in the workspace for which JVMPI needs to be enabled.
3. Click the **Process Definition** entry in the Additional Properties pane of the workspace.
4. Click the **Java Virtual Machine** link in the Additional Properties pane.
5. Type `-XrunpmiJvmpiProfiler` into the Generic JVM arguments field as shown in Figure 16-6 on page 703. Add this entry before or after any existing arguments in case you have other arguments already.

Additional Properties		
Debug Mode	<input type="checkbox"/>	specify arguments when HProf profiler support is enabled. <small>[i] Specifies whether to use the JVM debug output. The default is not to enable debug mode support.</small>
Debug arguments	<input type="text" value="-Djava.compiler=NONE -Xdebug -Xnc"/>	<small>[i] Specifies command-line debug arguments to pass to the Java virtual machine that starts the application server process. You can specify arguments when Debug Mode is enabled.</small>
Generic JVM arguments	<input type="text" value="-XrunpmiJvmpiProfiler"/>	<small>[i] Additional command line arguments for the JVM.</small>
Executable JAR file name	<input type="text"/>	<small>[i] Specifies a full path name for an executable jar file that the Java virtual machine uses.</small>
Disable JIT	<input type="checkbox"/>	<small>[i] Configure the JVM such that the Just-in-Time (JIT) compiler is disabled.</small>
Operating system name	<input type="text"/>	<small>[i] Specifies JVM settings for a given operating system. When started, the process will use the JVM settings for the operating system of the node.</small>

Apply OK Reset Cancel

Figure 16-6 Enabling JVMPI

6. Click **Apply** or **OK** to apply the changes.
7. Click **Save** to store the changes in the WebSphere configuration.
8. Start the application server or Node Agent, or restart the application server or Node Agent if it is currently running.

Make sure you have set the instrumentation level in the PMI setting on the JVM module to MAX for the profiler counters to be updated. Also, refresh the Tivoli Performance Viewer if you are using it.

**Important:** Node Agents and application servers collect data in the JVMPI counters of the JVM module regardless of having the command-line argument specified. Be aware that collected data is not reliable until the `-XrunpmiJvmpiProfiler` argument is specified.

## Enabling JVMPI with the command line interface

To enable JVMPI profiling using the `wsadmin` command interface, perform these steps:

1. Start `wsadmin`.

2. Enter the following command at the prompt:

```
wsadmin> $AdminConfig modify \
(cells/net1Network/nodes/node1/servers/server1:server.xml#\
JavaVirtualMachine_1) {{genericJvmArguments -XrunpmiJvmpiProfiler}}
```
3. Save the configuration and start the application server or Node Agent, or restart the application server or Node Agent if it was already running for the change to take effect.

Figure 16-7 shows the Tivoli Performance Viewer output with JVMPi enabled.

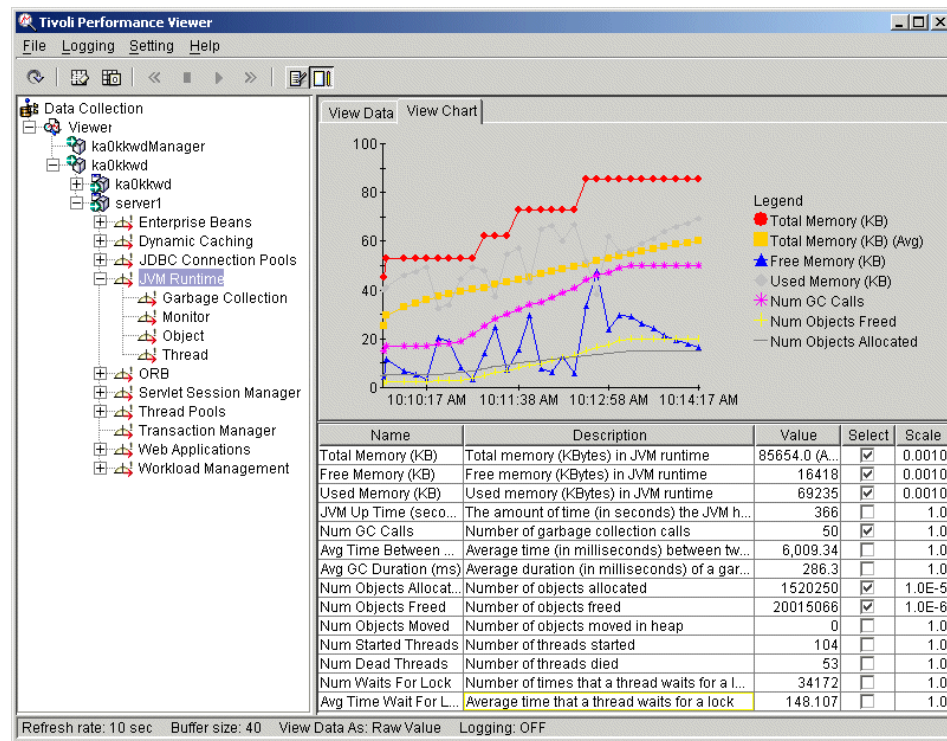


Figure 16-7 JVM module with JVMPi enabled

## Disabling JVMPi profiling

To disable the JVMPi profiling using the Administrative Console:

1. Follow the steps described in “Enabling JVMPi from the Administrative Console” on page 702 to get to the JVM properties for the server or Node Agent.
2. Remove the JVM command line argument `-XrunpmiJvmpiProfiler`.
3. Save your changes and restart your application server or Node Agent.



To disable the JVMPI profiling using wsadmin:

1. Use the **wsadmin** command shown in “Enabling JVMPI with the command line interface” on page 703, but change the `genericJvmArguments` to read:  

```
{{genericJvmArguments {}}}
```
2. Save the configuration and restart your application server or Node Agent.

## 16.2.8 Summary

Performance monitoring is an activity in which you collect and analyze data about the performance of your applications and their environment. We have just discussed the Performance Monitoring Infrastructure that provides the APIs to collect that performance data. This performance data can be monitored and analyzed with:

- ▶ Tivoli Performance Viewer (formerly known as Resource Analyzer), which is included in WebSphere Application Server
- ▶ Other IBM Tivoli monitoring tools sold separately
- ▶ User developed monitoring tools
- ▶ Third party monitoring tools

## 16.3 Using Tivoli Performance Viewer

Tivoli Performance Viewer is a real-time monitoring tool that displays PMI data. This tool ships as part of WebSphere Application Server (starting with version 4.0 for WebSphere Application Server for distributed platforms and with WebSphere Application Server for z/OS version 5.0). The tool can access data remotely and across platforms. This tool provides summary reports of key performance data and allows the user to view the data in tabular form or in graphical form. It can also record the information it collects and replay it in a log file without connecting to an application server.

The tool was formerly known as Resource Analyzer, but was renamed for the WebSphere Application Server V5 release.

### 16.3.1 About Tivoli Performance Viewer

The Tivoli Performance Viewer retrieves performance data by periodically polling the PMI service of the application server that is being monitored. As all application servers host a PMI service, only one connection is created from the Tivoli Performance Viewer to the Deployment Manager in a cell, and PMI data from all servers flow through the Deployment Manager to the TPV. In addition,

the Node Agent in a Network Deployment environment also hosts a PMI service for monitoring the running state of the physical machine. The performance data requested by Tivoli Performance Viewer is provided by the PMI service component by means of the Perf MBean residing in the application server's MBean server.

To minimize the performance impact, Tivoli Performance Viewer polls the server with the PMI data at an interval set by the user. All data manipulations are done in the Tivoli Performance Viewer client, which can be run on a separate machine, further reducing the impact. The Tivoli Performance Viewer's GUI provides controls that enable you to choose the particular resources and counters to include in the view. There are table and chart views available. You can also store retrieved data in a log file while viewing the data. This log file can later be used for replaying the scenario. The log file can have different formats, the preferred format is XML.

Figure 16-8 shows a high-level overview of how the Tivoli Performance Viewer collects data.

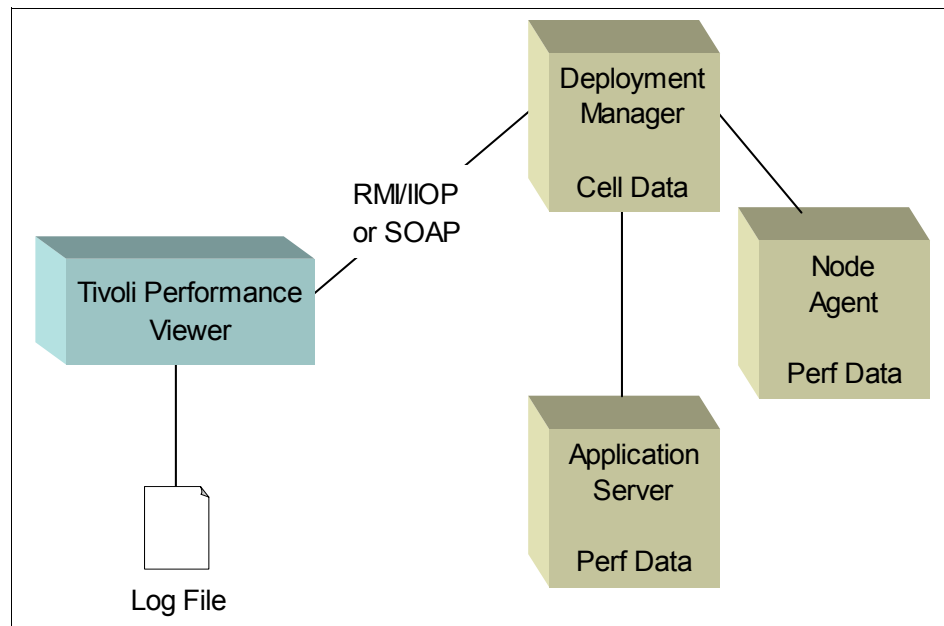


Figure 16-8 Tivoli Performance Viewer infrastructure overview

### 16.3.2 What can Tivoli Performance Viewer do?

The Tivoli Performance Viewer provides access to a wide range of performance data for three kinds of resources:

- ▶ Application resources (for example, servlet response time and EJB response time).
- ▶ WebSphere runtime resources (for example, application server thread pools and database connection pools).
- ▶ System resources (for example, CPU utilization).

Performance data includes simple counters, statistical data (such as the response time for each method invocation of an enterprise bean), and load data (such as the average size of a database connection pool during a specified time interval). This data is reported for individual resources and aggregated for multiple resources. See 16.2, “Performance Monitoring Infrastructure” on page 687 for more details about performance data organization.

#### **Tivoli Performance Viewer functionality**

Depending on which aspects of performance are being measured, you can use the Tivoli Performance Viewer to accomplish the following tasks:

- ▶ View data in real time or view historical data from log files.
- ▶ View data in chart form, allowing comparisons of one or more statistical values for a given resource on the same chart. In addition, different units of measurement can be scaled to enable meaningful graphic displays.
- ▶ Record current performance data in a log and replay performance data from previous sessions.
- ▶ Compare data for a single resource to an aggregate (group) of resources on a single node.

Given all this data, the Tivoli Performance Viewer can be used to do the following types of analysis:

- ▶ Monitor real-time performance, such as response times for servlet requests or enterprise bean methods.
- ▶ Detect trends by analyzing logs of data over short periods of time.
- ▶ Determine the efficiency of a configuration of resources (such as the amount of allocated memory, the size of database connection pools, and the size of a cache for enterprise bean objects).
- ▶ Gauge the load on application servers and the average response time for clients.

### 16.3.3 Running Tivoli Performance Viewer

To start Tivoli Performance Viewer on Windows platforms, do one of the following:

- ▶ Select **Start -> Programs -> IBM WebSphere -> Application Server v5.1 -> Tivoli Performance Viewer**.
- ▶ Or use the command line:  

```
<WAS_HOME>\bin\tpervviewer.bat <host_name> <port_number>  
<connector_type>
```

For example:

```
<WAS_HOME>\bin\tpervviewer.bat localhost 8879 SOAP
```

**Note:** On UNIX platforms, type into a shell:

```
<WAS_HOME>/bin/tpervviewer.sh <host_name> <port_number> <connector_type>
```

The default port number varies based on the WebSphere software version in use. For example, the default SOAP connector port for WebSphere Network Deployment is 8879 while the default port for WebSphere Base is 8880. The reason for this is that in a Network Deployment environment, TPV connects to the Deployment Manager host rather than to the individual application server.

If you start `tpervviewer.bat(sh)` *without* any parameters, it will automatically use the default port first. If that fails, a pop-up window asking for new ports appears.

#### ***Default SOAP connector ports***

- ▶ 8879 for WebSphere Application Server Network Deployment
- ▶ 8880 for WebSphere Application Server Base

#### ***Default RMI connector ports***

- ▶ 9809 for WebSphere Application Server Network Deployment
- ▶ 2809 for WebSphere Application Server Base

**Note:** In case your environment is not set up with the default ports, you can find the correct port numbers for your environment by looking for them using the Administrative Console or in `SystemOut.log` file.

**Tip for iSeries users:** On iSeries, you can connect the Tivoli Performance Viewer to an iSeries instance from either a Windows, an AIX, or a UNIX client as described above. To discover the RMI or SOAP port for the iSeries instance, start Qshell and enter the following command:

```
<product_installation_directory>/bin/dspwasinst -instance <myInstance>
```

Where:

- ▶ <product\_installation\_directory> is your iSeries install directory.
- ▶ <myInstance> is the instance name used when you created the iSeries instance.

## Connecting to the Deployment Manager

The Tivoli Performance Viewer will attempt to access the Deployment Manager on the local machine. By default, as shown Figure 16-9, it will use these settings:

host_name	localhost
port_number	8879
conn_type	SOAP

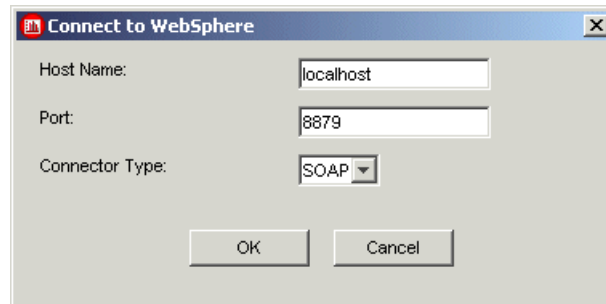


Figure 16-9 Setting Tivoli Performance Viewer connection parameters

## Running disconnected

If a connection cannot be established you will get a window that allows you to enter new connection properties. Clicking **Cancel** from this window starts Tivoli Performance Viewer in a disconnected state. In this state, you can replay a previously recorded session (see “Recording in a log file” on page 719).

## Accessing a remote server

If you want to access a remote machine running the Deployment Manager, Node Agent or an application server, the command syntax is:

```
tperfvier.bat|sh [host_name [port_number [conn_type ]]]
```

Where:

host_name	Is the host name or IP address of the remote system (the default is localhost).
port_number	Is the port number of the remote system (the default is 8879).
conn_type	Is the type of connection, either SOAP or RMI (the default is SOAP).

For example, if you want to access a remote UNIX machine running the Node Agent from your Windows system, type in the following:

```
tperfvviewer.bat|sh <mywas.mydomain.com> 8879 SOAP
```

If the default port 8879 and SOAP protocol are used, they can be left out.

After starting the Tivoli Performance Viewer successfully, you get the main window shown in Figure 16-10 on page 710. The Tivoli Performance Viewer window consists of the menu bar at the top, the toolbar just below the menu bar, on the left-hand side the Resource Selection pane, on the right-hand side the data monitoring pane (can contain table and charts alternatively), and on the bottom the status bar.

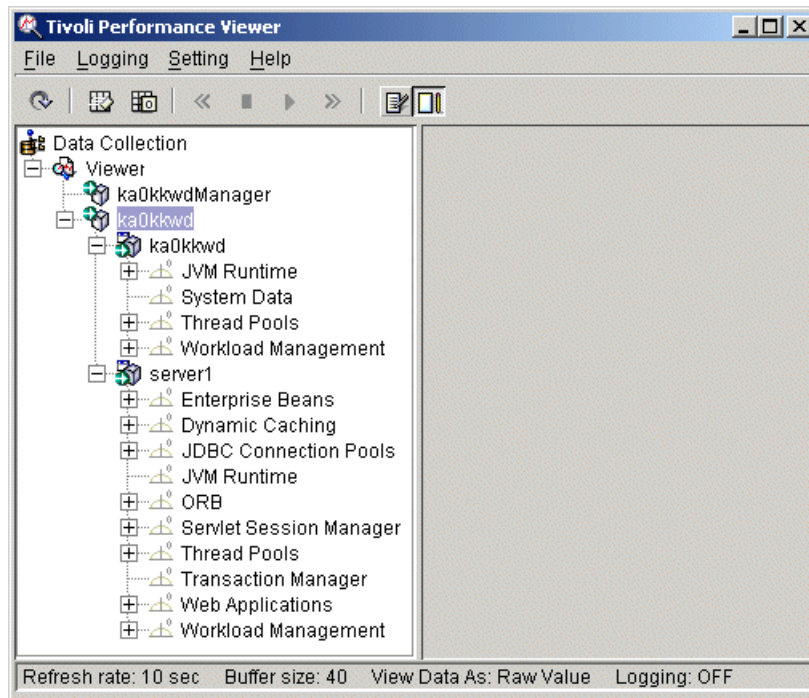


Figure 16-10 Tivoli Performance Viewer

## Stopping the Tivoli Performance Viewer

Stop the Tivoli Performance Viewer by selecting **File** -> **Exit** from the main window. This will bring up a window if PMI monitoring levels have been changed through TPV's interface. You are asked whether you want to save the monitoring settings in the Admin Server or not. Choosing **Yes** saves the performance monitoring settings within the application server; choosing **No** discards them.

## Tivoli Performance Viewer with security enabled

You can use Tivoli Performance Viewer also for WebSphere Application Servers that are configured for global security. However, some extra configuration is needed in this case. The configuration depends on the connector protocol you are using (SOAP or RMI).

### *For the SOAP connector*

In order to run a Tivoli Performance Viewer client application with security enabled, you must have %CLIENTSOAP% and %CLIENTSAS% properties on your Java Virtual Machine command line. The %CLIENTSOAP% and %CLIENTSAS% properties are defined in the setupCmdLine.bat or setupCmdLine.sh files. Do the following:

1. Set com.ibm.SOAP.securityEnabled to **True** in the soap.client.props file for the SOAP connector.

The soap.client.props property file is located in the <WAS\_ROOT>/properties directory.

2. Set com.ibm.SOAP.loginUserid and com.ibm.SOAP.loginPassword as the user ID and password for login.

**Tip:** A common mistake is to leave extra spaces at the end of the lines in the property file. Do not leave extra spaces at the end of the lines, especially for the user ID and password lines.

### *For the RMI connector*

If you are using the Remote Method Invocation (RMI) connector to access the remote server, then you get a user ID/password challenge window, as shown in Figure 16-11, before the main window. At this point, enter a valid user ID and password and click **OK** to log on.

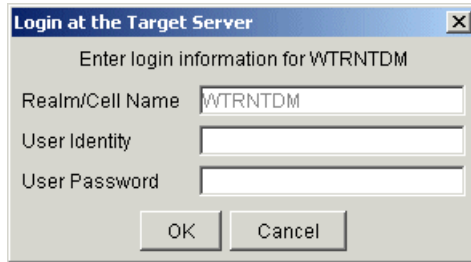


Figure 16-11 Tivoli Performance Viewer login challenge

Instead of entering the user\_ID/password in the pop-up window every time, you can set these values in the `sas.client.props` file.

### 16.3.4 Setting the instrumentation levels

On the surface, setting the instrumentation levels with the Tivoli Performance Viewer is similar to setting the levels with the Administrative Console. However, you will see that if you opt to customize the settings you can do so at a more granular level using the Tivoli Performance Viewer.

1. To open the Performance Monitoring Settings window, click **Data Collection** in the left pane of the Tivoli Performance Viewer main window.
2. Select the **Current Activity** radio button in the right pane. Alternatively, select **File -> Current Activity** to view the monitoring settings.



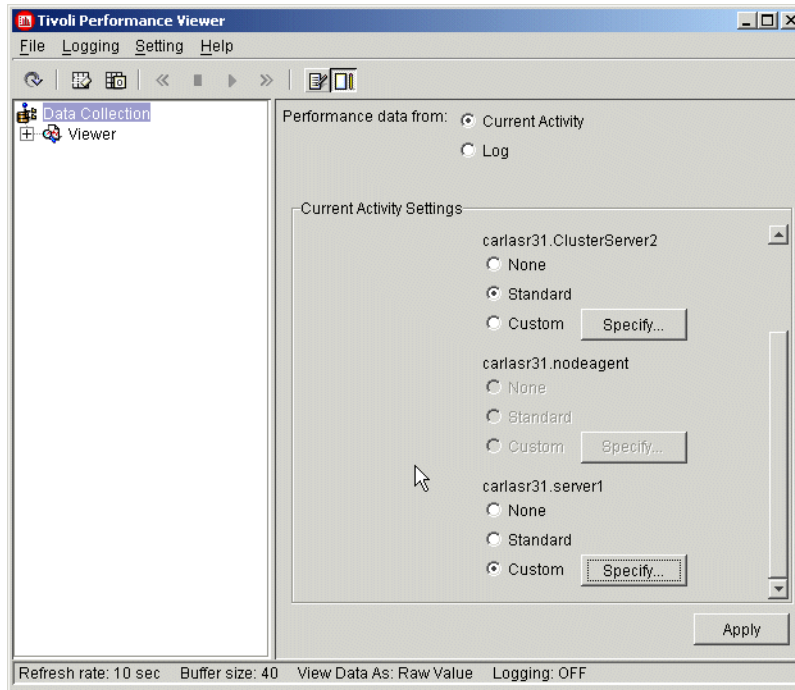


Figure 16-12 Setting the instrumentation levels

3. Select the setting for each server. You can choose **None**, **Standard**, or **Custom**. To customize the settings:
  - a. Select **Custom** and click the **Specify...** button.
  - b. Select a resource category in the hierarchy and select the required monitoring level on the right.
  - c. Click **OK** to close the window.
4. When done, click **Apply**.

## Custom settings

The Tivoli Performance Viewer allows you to set instrumentation levels using a graphical view of the modules and submodules. As you select individual application elements and set the level, a list of the counters associated with that level is displayed. The Administrative Console offers instrumentation level access only on the top-level modules.

Changing or setting the instrumentation level always starts the performance data reporting immediately for the selected resource category.

The counters available for the selected resource category will be shown on the lower right (in the Counters box) when you choose the monitoring level. Figure 16-13 shows the counters available for Transaction Manager with the instrumentation level set to **Low**.

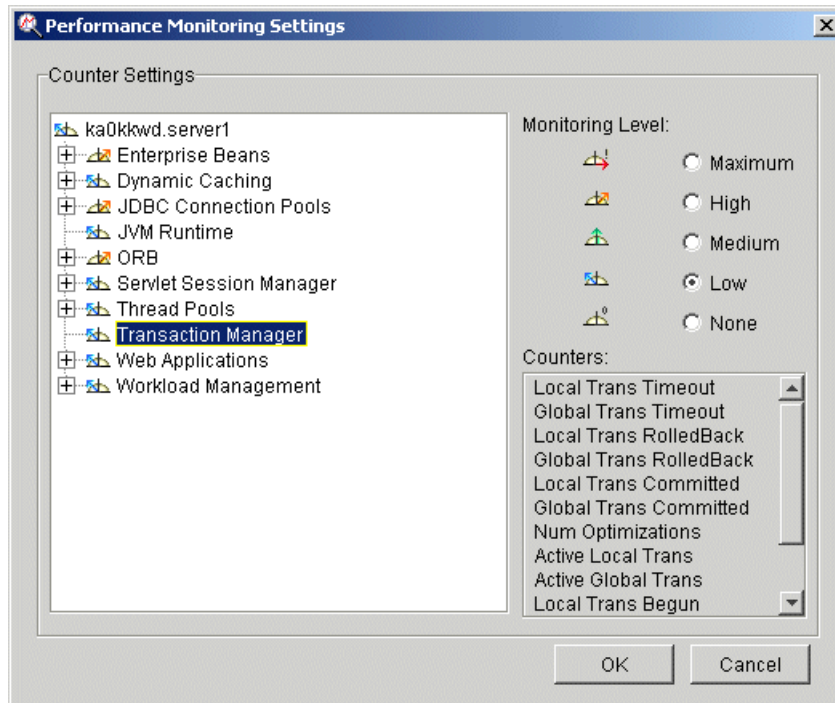


Figure 16-13 Performance Monitoring Settings for Transaction Manager

Setting a particular instrumentation level for one resource category applies the same instrumentation level to all of its subcategories. You can specify a different level for an individual resource subcategory by selecting it and specifying the desired level.

Expand the resource categories to make sure that only the desired level is applied; otherwise the performance impact may be higher than you expected.

**Important:** Instrumentation level settings are permanent even after restarting the application server or Node Agent. You have to disable them explicitly.

## Setting instrumentation level for enterprise bean methods

To avoid the overhead of monitoring individual remote methods, individual methods in enterprise beans will not be displayed in the performance pane unless the monitoring level for Methods is set to maximum. To display individual methods and specify their instrumentation levels, do the following:

1. Select **Custom** for the server and click **Specify...** (see Figure 16-12 on page 713).
2. Expand **Enterprise Beans**, then expand the JAR file for the EJB.
3. Expand the appropriate category (stateful session bean, entity bean, etc.).
4. Expand the bean and select the **Methods** category. Set the instrumentation level to **Maximum**.

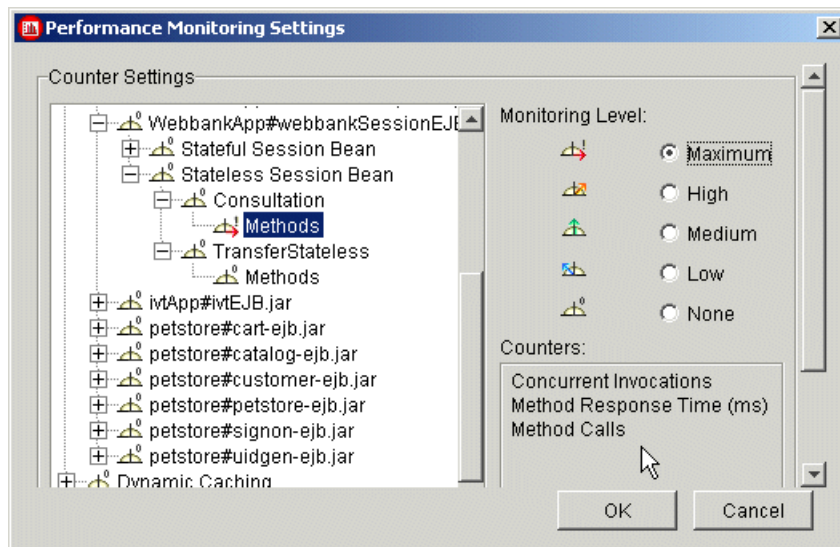


Figure 16-14 Set the instrumentation level to maximum for the enterprise bean methods

5. Click **OK** to close the window.
6. Click **Apply** to make the change take effect. Then re-open the Performance Monitoring Settings window (select **Custom** and click **Specify...**). Individual methods are now displayed as shown in Figure 16-15 on page 716.

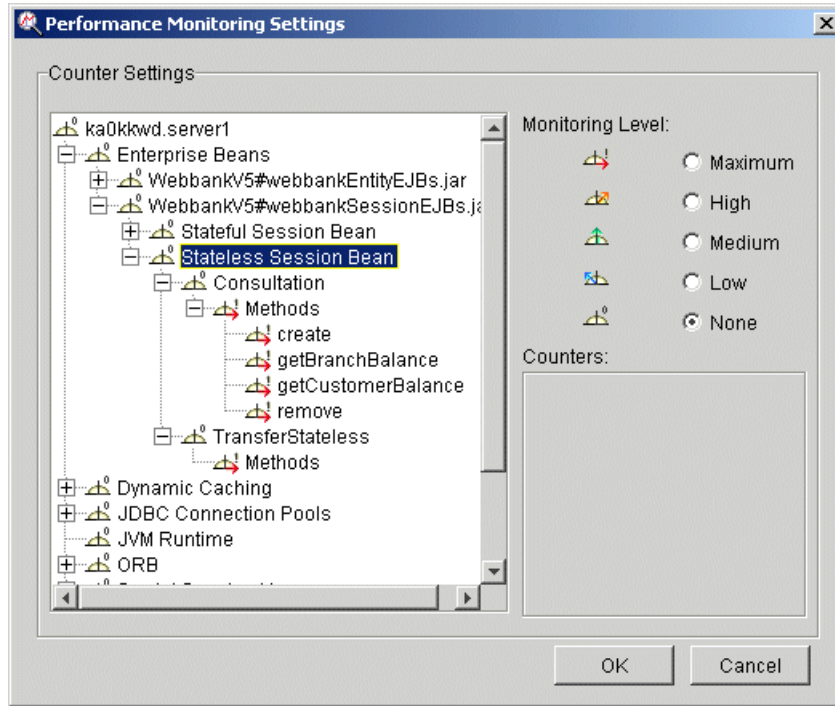


Figure 16-15 Enterprise bean methods available after the first method call

7. Select the individual methods and set the instrumentation levels as required.

Note that only methods that have been called by an application are displayed. If a remote method has not been called since the application server was started, it does not appear in the performance pane.

In Figure 16-15, for an example, the Consultation enterprise bean has been called at least once, but the TransferStateless bean has not been called yet.

### 16.3.5 Using Tivoli Performance Viewer to monitor an application

In this section we explain the steps we used to monitor one of the sample enterprise applications installed into the WebSphere administrative domain by default.

Before starting the Tivoli Performance Viewer, the PMI service has to be enabled for the application servers you would like to monitor. See 16.2.5, “Enabling the PMI service” on page 697 for a description of how to enable the PMI service.

## Refreshing data

New performance data can become available in either of the following situations:

- ▶ An administrator uses the console to change the instrumentation level for a resource (for example, from medium to high).
- ▶ An administrator uses the console to add a new resource (for example, an enterprise bean or a servlet) to the run time.

In both cases, if the resource in question is already polled by the Tivoli Performance Viewer or the parent of the resource is being polled, the system is automatically refreshed. If more counters are added for a group that the Performance Viewer is already polling, the Performance Viewer automatically adds the counters to the table or chart views. If the parent of the newly added resource is polled, the new resource is detected automatically and added to the Resource Selection tree. You can refresh the Resource Selection tree, or parts of it, by selecting the appropriate node and clicking the Refresh icon, or by right-clicking a resource and choosing Refresh.

When an application server runs, the Performance Viewer tree automatically updates the server local structure, including its containers and enterprise beans, to reflect changes on the server. However, if a stopped server starts after the Performance Viewer starts, a *manual* refresh operation is required so that the server structure accurately reflects in the Resource Selection tree.

Clicking Refresh with server selected under the **Viewer** icon causes TPV to query the server for new PMI and product configuration information.

Use the following steps:

1. Select one or more resources in the Resource Selection panel.
2. Click **File -> Refresh**. Alternatively, click the **Refresh** icon or right-click the resource and select **Refresh**.

Clicking refresh with server selected under the **Advisor** icon causes TPV to refresh the advice provided, but will not refresh PMI or product configuration information.

## Displaying multiple counters

To analyze the performance data captured by Tivoli Performance Viewer, it is often necessary to view counters from multiple modules. To select multiple counters in one chart or table:

1. Hold down the Ctrl key and select the resource categories whose counters you want to see.

2. Select the required counters using the Select column in the counter selection window on the lower left.

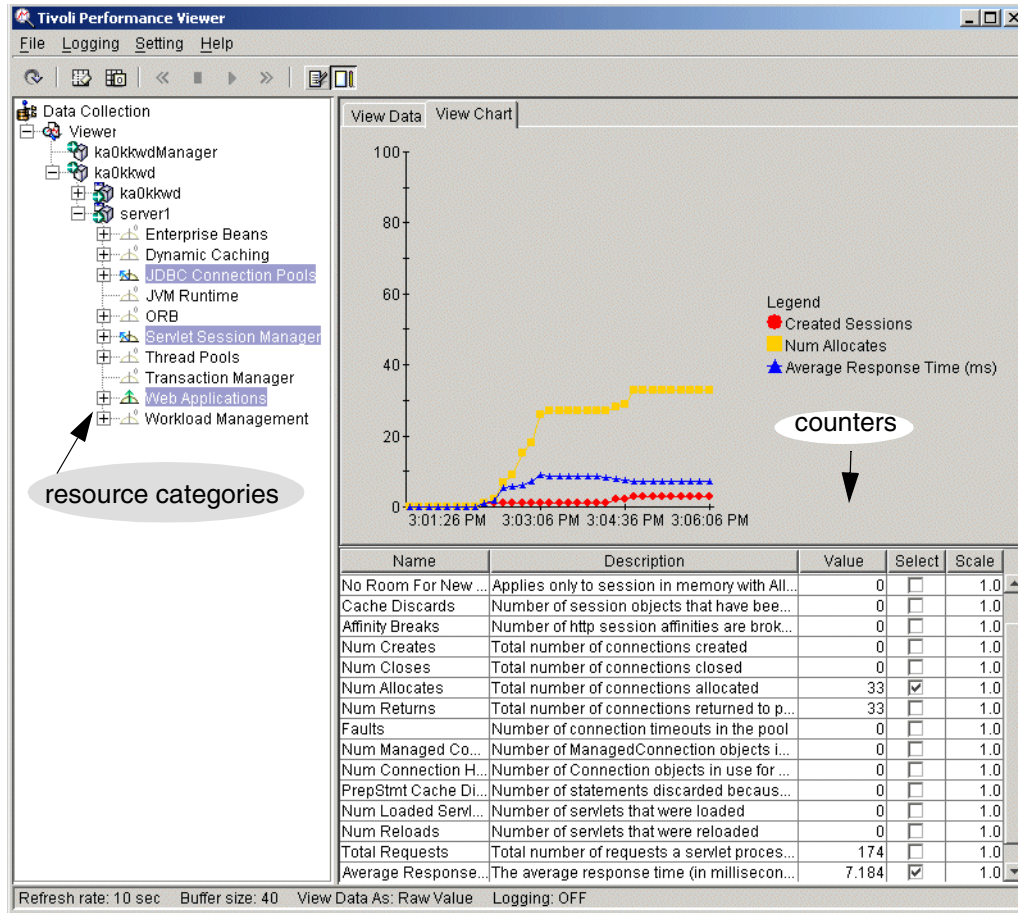


Figure 16-16 Selecting multiple counters from different resource categories

For example, in Figure 16-16, you can see that the counters for Created Sessions (from Servlet Session Manager), Num Allocates (from JDBC Connection Pools), and Average Response Time (from Web Applications) have all been selected.

## Recording in a log file

All data being reported by the Tivoli Performance Viewer can be saved in a log file. The data is written to the log as serialized Java objects or as an XML document. To start recording data, do the following:

1. Select **Logging -> On...** from the Tivoli Performance Viewer main menu or click the **Logging on/off** icon in the toolbar.
2. In the Save log file window, specify the file name (and path), and select **Save as type**. The Save as type field allows an extension of \*.perf (Tivoli Performance Viewer logs) for binary files or \*.xml for XML files. XML is the recommended format.
3. Click **OK**.

To stop the recording of data, select **Logging -> Off** from the main menu or click the **Logging on/off** icon in the toolbar.

## Replaying a log file

Both log file types can be replayed using the Tivoli Performance Viewer. If size is of concern, it is recommended that you zip the files while they are not needed. To replay a log file, do the following:

1. Select **File -> Log** from the Tivoli Performance Viewer main menu.
2. Click the **Browse...** button to open the file browser.
3. In the Open window, locate the name of the file to replay and click the **Open** button.
4. Click the **Play** icon from the toolbar or select **Setting -> Log Replay -> Play** from the main menu.

By default, the data is replayed at the same rate it is collected (written to the log). If data is collected every minute, it is displayed every minute. You can change the speed at which the log is replayed by clicking the **FF** button (Fast Forward) in the toolbar or by selecting **Setting -> Log Replay -> FF** from the main menu.

While replaying the log, you can change your resource selections with the resource selection pane. You can also view the data in either of the views available (data or chart) in the data display window.

You can stop and resume the log at any point. However, data cannot be replayed in reverse.

To rewind the log file, first you must click **Stop**, then click the **Rewind** button in the toolbar or select **Setting -> Log Replay -> Rewind** from the main menu.

## Clearing values from tables and charts

After stopping a resource, use the **Clear Values** operation to remove the remaining data from a table or chart. You can then begin populating the table or chart with new data. To clear the values currently displayed, do the following:

1. Select one or more resources in the resource hierarchy.
2. Select **Setting** -> **Clear Buffer** from the main menu. Alternatively, click the **Clear Buffer** icon in the toolbar.

## Resetting counters to zero

To reset the start time for calculating aggregate data, do the following:

1. Select one or more resources in the resource hierarchy.
2. Select **Setting** -> **Reset to Zero** from the main menu. Alternatively, click the **Reset to Zero** icon in the toolbar.

The reset operation sets the clock used for reporting aggregate data for counters of the selected performance category. Instead of reporting data from the time the server was started, reporting now begins from the time of the reset action. Not all counters can be reset. If you use the reset operation for a group containing counters that cannot be reset, the reset action has no effect. You can select multiple performance groups and reset them simultaneously.

**Note:** The clear and reset operations are applicable to both real-time mode and log replay mode.

## Viewing and modifying chart data

When selected counters are using measurement units that are not proportionally similar, the scaling factor can be set manually to allow a more meaningful display. The following sections explain how you can manually change the scaling factor for the chart view.

### *Scaling the chart display manually*

To manually change the scale of a counter:

1. In the counter selection window, on the lower right, double-click the **Scale** column for the counter that you want to modify.
2. Type the desired scale value for the counter into the field.

The chart view will be updated immediately to reflect the change in the scaling factor.



The possible values for the Scale field range from 0 to 100 and show the following relationships:

- < 1      Scaling reduces the value. For example, a scale of 0.5 means that the data points for the variable are reduced to half of their actual values.
- = 1      The value is not scaled. This is the default.
- > 1      Scaling increases the value. For example, a scale of 1.5 means that the data points for the variable are increased to one and one-half times their actual values.

Negative results are displayed as zero (0).

This value is reflected only in the View Chart window.

## Changing display settings

This section looks at some of the Tivoli Performance Viewer display settings:

- ▶ Specifying a different refresh rate
- ▶ Changing the data view
- ▶ Changing the buffer size

### ***Specifying a different refresh rate***

By default, the Tivoli Performance Viewer retrieves data from the administrative server every 10 seconds. To change the rate at which data is retrieved from the server, do the following:

1. Select **Setting** -> **Set Refresh Rate...** from the main menu.
2. In the Set Refresh Rate window, type a positive integer representing the number of seconds. The integer must be 1 or greater.
3. Click **OK**.

### ***Changing the data view***

The data view mode determines whether counter values represent absolute values, changes in values, or rates of change. The data view mode meanings differ slightly depending on where you are viewing data. To change the data view mode:

1. Select **Setting** -> **View Data As** from the main menu.
2. Select from the following choices:
  - **Raw Value.** Displays the absolute values. If the counter represents load data (for example, the pool size or the average number of live bean objects), the Tivoli Performance Viewer displays the current value followed by the average, for example 18(avg:5).

- **Change in Value.** Displays the change in the current value from a previous value.
- **Rate of Change.** Displays the ratio  $\text{change}/(T1 - T2)$ , where change is the change in the current value from a previous value, T1 is the time when the current value was retrieved and T2 is the time when the previous value was retrieved.

### ***Changing the buffer size***

By default, the View Data window displays 40 rows, corresponding to the values of the last 40 data points retrieved from the server. To change the size of the table (number of rows displayed):

1. Select **Setting -> Set Buffer Size...** from the main menu.
2. In the Set Buffer Size window, specify the number of rows to display.
3. Click **OK**.

## **16.3.6 Getting online help**

WebSphere Tivoli Performance Viewer has HTML-based online help. To see the help pages, select **Help -> Help Topics** from the main menu. There are also a number of Tivoli Performance Viewer articles in the WebSphere InfoCenter, available at:

<http://www.ibm.com/software/webservers/appserv/infocenter.html>

## **16.4 Other performance monitoring and management solutions**

In addition to the tools covered in the previous sections of this chapter, IBM and third-party vendors offer additional performance monitoring and management tools. However, these tools have to be purchased separately.

For example, Tivoli offers IBM Tivoli Monitoring for Web Infrastructure and IBM Tivoli Monitoring for Transaction Performance. For more information, see:

<http://www.ibm.com/software/tivoli/products/monitor-web/>

<http://www.ibm.com/software/tivoli/products/monitor-transaction/>

Also, several other companies provide performance monitoring, problem determination, and management solutions that can be used with WebSphere Application Server. These products use WebSphere Application Server interfaces, including Performance Monitoring Infrastructure (PMI), Java

Management Extensions (JMX), and PMI Request Metrics Application Response Measurement (ARM).

Use the following URL to find a list of IBM Business Partners that offer performance monitoring tools compliant with WebSphere Application Server:

[http://www.ibm.com/software/webserver/pw/dhtml/wsperformance/performance\\_bpsolutions.html](http://www.ibm.com/software/webserver/pw/dhtml/wsperformance/performance_bpsolutions.html)

## 16.5 Developing your own monitoring application

If for some reason you would like to develop your own monitoring application, the PMI offers three interfaces for you:

- ▶ The Java Management Extension (JMX) interface

The JMX interface is accessible through the AdminClient tool.

- ▶ PMI client interface

The PMI client interface is a Java interface that works with Version 3.5.5 and above.

- ▶ A servlet interface

The servlet interface is perhaps the simplest, requiring minimal programming, as the output is XML.

For detailed information on how to implement your own monitoring application using one of these two interfaces, refer to the IBM WebSphere Application Server V5.1 InfoCenter. Search for “Developing your own monitoring applications”, this will bring you to the appropriate articles in the InfoCenter.

## 16.6 PMI Request Metrics

Request Metrics is a tool that allows you to track individual transactions, recording the processing time in each of the major WebSphere Application Server components. The information tracked may either be saved to log files for later retrieval and analysis, be sent to ARM Agents, or both. Request Metrics is different from PMI instrumentation in that PMI provides the aggregation and averages across all the transactions (such as average servlet response time across transaction A, B, and C), while Request Metrics provides response time for each individual transaction (such as the servlet response time for transaction A.)

As a transaction flows through the system, Request Metrics tracks on additional information so that the log records from each component can be correlated, building up a complete picture of that transaction. The result looks similar to the following:

```
HTTP request /trade/scenario -----> 172 ms
Servlet/trade/scenario -----> 130 ms
    EJB    TradeEJB.getAccountData    --> 38 ms
        JDBC select    -> 7 ms
```

This transaction flow with associated response times can help users target performance problem areas and debug resource constraint problems. For example, the flow can help determine if a transaction is spending most of its time in the Web server plug-in, the Web container, the enterprise bean container or the backend database. The response time collected for each level includes the time spent at that level and the time spent in the lower levels. In the example above, the response time for the servlet, which is 130 milliseconds, also includes 38 milliseconds from the EJB and JDBC. Therefore, 92 ms can be attributed to the servlet process.

Request Metrics tracks the response time for a desired transaction. For example, tools can inject synthetic transactions. Request Metrics can then track the response time within the WebSphere environment for those transactions. A synthetic transaction is one that is injected into the system by administrators in order to proactively test the performance of the system. This information can help administrators tune the performance of the Website and take corrective actions should they be needed. Thus, the information provided by Request Metrics might be used as an alert mechanism to detect when the performance of particular request type goes beyond acceptable thresholds. The filtering mechanism within Request Metrics may be used to focus on the specific synthetic transactions and can help optimize performance in this scenario.

Three types of filters are supported:

- ▶ Originator IP filter
- ▶ URI filter
- ▶ EJB method name filter

When filtering is enabled, only requests matching the filter generate request metrics data, create log records, and/or call the ARM interfaces. This allows work to be injected into a running system (specifically to generate trace information) to evaluate the performance of specific types of requests in the context of normal load, ignoring requests from other sources that might be hitting the system.

## 16.6.1 Enabling and configuring PMI Request Metrics

Request Metrics are enabled and configured globally for all application servers in the cell. For a single-server environment, the metrics are enabled and configured for all application servers residing on the local node.

The PMI Request Metrics component can be enabled, disabled, and configured at runtime, without having to restart each individual application server in the cell.

To enable the PMI Request Metrics using the Administrative Console:

1. Select **Troubleshooting** -> **PMI Request Metrics**. From here you can do the following:
  - Enable/disable the Request Metrics.
  - Enable an ARM agent. This implies that you have installed an ARM agent that supports Request Metrics.
  - Specify the level of detail on the Request Metrics data, as shown in Figure 16-17. A setting of NONE produces no metrics data. HOPS shows the EJB method invoked, requested URI, or JDBC statement executed, along with the response time. PERF\_DEBUG and DEBUG give you more detailed information but are also more performance intensive.

The screenshot shows the 'Performance Monitoring Request Metrics' configuration pane. At the top, it says 'Specify Performance Monitoring Request Metric configuration.' followed by a help icon. Below this is a 'Configuration' tab. Under the 'General Properties' section, there are three rows: 'Request Metrics' with a checked 'Enable' checkbox and a help icon; 'Application Response Measurement (ARM)' with an unchecked 'Enable ARM' checkbox and a help icon; and 'Trace Level' with a dropdown menu set to 'HOPS' and a help icon. Below these rows are buttons for 'Apply', 'OK', 'Reset', and 'Cancel'. At the bottom, there is an 'Additional Properties' section with a 'Filters' link and the text 'A set of request metric filters.'

Performance Monitoring Request Metrics		
Specify Performance Monitoring Request Metric configuration. <a href="#">?</a>		
<b>Configuration</b>		
<b>General Properties</b>		
Request Metrics	<input checked="" type="checkbox"/> Enable	<a href="#">?</a> Enables Request Metrics.
Application Response Measurement (ARM)	<input type="checkbox"/> Enable ARM	<a href="#">?</a> Enables Performance Monitoring Infrastructure Request Metrics to call an underlying Application Response Measurement (ARM) agent.
Trace Level	<input type="text" value="HOPS"/>	<a href="#">?</a> Specifies how much trace data to accumulate for a given request.
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>		
<b>Additional Properties</b>		
<a href="#">Filters</a> A set of request metric filters.		

Figure 16-17 PMI Request Metrics configuration pane

As soon as Request Metrics are enabled and a trace level greater than NONE is specified and saved to the WebSphere configuration, trace records are written to the System.out JVM log for all application servers for any incoming Web or EJB request.

See the next section for a detailed description of the trace record format.

**Important:** In version 5.1, only remote EJB calls entering the RMI activity service are monitored by the PMI Request Metrics component. Hence a servlet calling a local EJB does not produce any metrics data for the bean.

In future releases, different tracing levels will give more instrumentation.

## 16.6.2 PMI Request Metrics trace record format

The trace records for Performance Monitoring Infrastructure (PMI) Request Metrics data are output to two log files: the Web server plug-in log file and the application server log file. The default directory for these log files is <\$WAS\_ROOT/logs> (or the name given to your server <\$WAS\_ROOT/logs/server\_name>) and default names are SystemOut.log and http\_plugin.log. Users might, however, specify these log file names and their locations.

In the WebSphere Application Server log file the trace record format is:

```
PMRM0003I:
parent:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
- current:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
  type=TTT detail=some_detail_information elapsed=nnnn
```

In the Web server plug-in log file the trace record format is:

```
PLUGIN:
parent:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
- current:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
  type=TTT detail=some_detail_information elapsed=nnnn bytesIn=nnnn
  bytesOut=nnnn
```

The trace record format is composed of two correlators: a parent correlator and current correlator. The parent correlator represents the upstream request and the current correlator represents the current operation. If the parent and current correlators are the same, then the record represents an operation that occurred as it entered WebSphere Application Server.

To correlate trace records for a particular request, collect records with a message ID of PMRM0003I from the appropriate application server log files and the PLUGIN trace record from the Web server plug-in log file. Records are correlated by matching current correlators to parent correlators. The logical tree can be created by connecting the current correlators of parent trace records to the parent correlators of child records. This tree shows the progression of the request across the server cluster. Refer to the article *Measuring data requests*

(*Performance Monitoring Infrastructure Request Metrics*) in the InfoCenter for an example of the transaction flow.

The parent correlator is denoted by the comma separating fields following the keyword "parent:". Likewise, the current correlator is denoted by the comma separating fields following "current:".

The fields of both parent and current correlators are listed in Table 16-3:

*Table 16-3 Request Metrics: Parent and current correlators*

Field	Description
ver	The version of the correlator. For convenience, it is duplicated in both the parent and current correlators.
ip	The IP address of the node of the application server that generated the correlator.
pid	The process ID of the application server that generated the correlator.
time	The start time of the application server process that generated the correlator.
reqid	An ID assigned to the request by Request Metrics, unique to the application server process.
event	An event ID assigned to differentiate the actual trace events.

Following the parent and current correlators, is the metrics data for timed operation (see Table 16-4):

*Table 16-4 Request Metrics: Metrics data for timed operation*

Field	Description
type	A code representing the type of operation being timed. Supported types include URI, EJB and JDBC. (See the description of Universal Resource Identifier (URI), Enterprise bean and Java Database Connectivity (JDBC) below.)
detail	Identifies the name of the operation being timed.
elapsed	The measured elapsed time in <units> for this operation, which includes all sub-operations called by this operation. The unit of elapsed time is milliseconds.
bytesIn	The number of bytes from the request received by the Web server plug-in.
bytesOut	The number of bytes from the reply sent from the Web server plug-in to the client.

The type and detail fields are described in Table 16-5.

Table 16-5 Request Metrics: Type and data fields

Field	Description
HTTP	The Web server plug-in generates the trace record. The detail is the name of the URI used to invoke the request.
URI	The trace record was generated by a Web component. The URI is the name of the URI used to invoke the request.
EJB	The fully qualified package and method name of the enterprise bean.
JDBC	The values select, update, insert or delete for prepared statements. For non-prepared statements, the full statement can appear.

Examples of URI and EJB Request Metrics records are shown in Example 16-2 and Example 16-3.

Example 16-2 URI Request Metrics

```
[11/29/03 9:46:31:578 EST] 7bca033b PmiRmArmWrapp I PMRM0003I:
parent:ver=1,ip=192.168.0.1,time=1016556185102,pid=884,reqid=4096,event=1
- current:ver=1,ip=192.168.0.1,time=1016556186102,pid=884,reqid=4096,event=1
type=URI detail=/hitcount elapsed=60
```

Example 16-3 EJB (EJB is the child of the servlet request in the downstream process)

```
[11/29/03 9:00:16:812 EST] 6d23ff00 PmiRmArmWrapp I PMRM0003I:
parent:ver=1,ip=192.168.0.1,time=1016556186102,pid=884,reqid=4096,event=1
- current:ver=1,ip=192.168.0.2,time=1016556122505,pid=9321,reqid=8193,event=1
type=EJB detail=com.ibm.defaultapplication.IncrementBean.increment elapsed=10
```

### 16.6.3 Filters

Filters are important for producing only the output necessary for a given monitoring task. A named URI, EJB, or client IP address can be specified and only the needed metrics data is produced. The performance load added to the containers by the Request Metrics component is reduced by adding more restrictive filters.

For HTTP requests arriving at a Web container, it is possible to filter on the URI and client IP address. For incoming requests to the EJB container, it is possible to filter on the bean method name and the client IP address. All the filters are listed and described here:

- **Client IP address filters:** Requests are filtered based on a known IP address. You can specify a mask for an IP address using the asterisk (\*). If



used, the asterisk must always be the last character of the mask, for example 127.0.0.\*, 127.0.\*, 127\*.

- ▶ **URI filters:** Requests are filtered based on the URI of the incoming HTTP request. The rules for pattern matching are the same as for matching client IP address filters.
- ▶ **Enterprise bean method name filters:** Requests are filtered based on the full name of the enterprise bean method. As with IP address and URI filters, you can use the asterisk (\*) to provide a mask. The asterisk must always be the last character of a filter pattern.
- ▶ **Filter combinations:** If both URI or EJB and client IP address filters are active, then Request Metrics require a match for both filter types. If neither is active, all requests are considered a match.

It is important to understand that filters are applied to requests as they enter the application server from the client (at the Web or EJB container level). Depending on defined filters, the request is either marked to have metrics generated or not marked to mean that no Request Metrics records should be produced for the duration of the request.

### 16.6.4 Example: Generating trace records from Performance Monitoring Infrastructure Request Metrics

Please refer to the WebSphere InfoCenter at:

<http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp>

Search for “Generating trace records from Performance Monitoring Infrastructure Request Metrics”. Here you will find an example on how use Request Metrics.

## 16.7 Dynamic Cache Monitor

J2EE applications have high read/write ratios and can tolerate small degrees of latency in the currency of their data. WebSphere Application Server consolidates several caching activities, including servlets, Web services, and WebSphere commands into one service called the *dynamic cache*. These caching activities work together to improve application performance, and share many configuration parameters, which are set in an application server's dynamic cache service.

Therefore the dynamic cache creates an opportunity for significant gains in server response time, throughput, and scalability. You can use the dynamic cache to improve the performance of servlet and JSP files by serving requests

from an in-memory cache. Cache entries contain servlet output, results of servlet execution, and metadata.

The Dynamic Cache Monitor is an installable Web application that displays simple cache statistics, cache entries and cache policy information. For detailed information, see Chapter 14, “Dynamic caching” on page 527.

## 16.8 Monitoring the IBM HTTP Server

The Windows version of IBM HTTP Server includes Windows performance monitor hooks. This allows you to use the Windows NT performance monitor or the Windows 2000 system monitor to observe the current state of an active IBM HTTP Server, along with all kinds of other system resources.

### 16.8.1 Configure your IBM HTTP Server

The default configuration of the IBM HTTP Server provides some performance data to the Windows performance monitoring tools. You can enable more performance data by modifying the HTTP server configuration file (`httpd.conf`). Add or uncomment the lines shown in Example 16-4 from the `httpd.conf` file and restart the IBM HTTP Server.

*Example 16-4 Enable extended performance data in `httpd.conf`*

---

```
LoadModule status_module modules/ApacheModuleStatus.dll
ExtendedStatus On
```

---

For more information about modifying the HTTP server configuration, refer to the IBM HTTP Server InfoCenter:

<http://www.ibm.com/software/webservers/httpservers/library.html>

### 16.8.2 Starting the Windows performance monitor

On Windows NT, start the performance monitor by selecting **Start -> Programs -> Administrative Tools (Common) -> Performance Monitor**.

On Windows 2000, use the following steps to start the system monitor:

1. Click **Start -> Settings -> Control Panel** to open the Control Panel window.
2. Double-click **Administrative Tools** to open the Administrative Tools window.
3. Double-click **Performance** to open the Performance window.
4. Select the **Tree** tab on the top-left and click **System Monitor** to activate it.

### 16.8.3 Selecting performance data

To select the IBM HTTP Server as the source for your performance data and specify the counters, do the following:

1. Click the plus icon on the taskbar to open the Add Counters window.
2. Select **IBM HTTP Server** from the Performance Object drop-down list.
3. From the Instance field, choose one or more process IDs (if there is only a 0 shown, this means no instance of IBM HTTP Server is running).
4. Go to the Counters field and select the required counter. Use the **Explain** button on the right-hand side to get a description of a counter. Select multiple counters by holding down the Shift or Ctrl key.
5. Click **Add** to add your selection to your monitor.
6. Click **Done** or **Close** and the data reporting starts.

Figure 16-18 shows an example of a monitoring session.

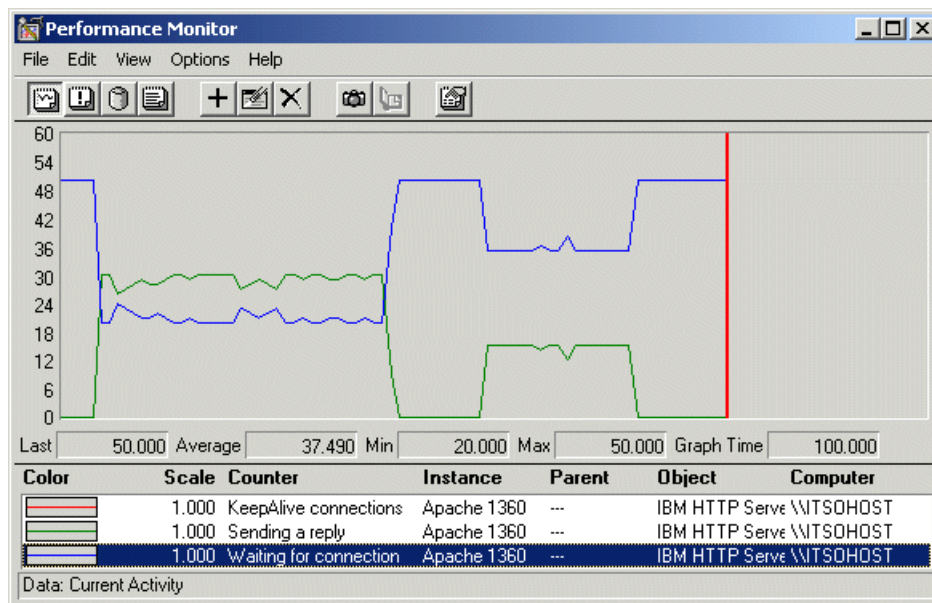


Figure 16-18 Windows NT performance monitor

### 16.8.4 IBM HTTP Server status page

The IBM HTTP Server server-status page is available on all supported IBM HTTP Server platforms. It shows performance data on a Web page in HTML format.

Perform the following steps to activate the server-status page:

1. Open the IBM HTTP Server file httpd.conf in an editor.
2. Remove the comment character "#" from the following lines:

```
#LoadModule status_module modules/ApacheModuleStatus.dll
#<Location /server-status>
#SetHandler server-status
#</Location>
```

3. Save the changes and restart the IBM HTTP Server.
4. Open the URL `http://<yourhost>/server-status` in a Web browser, and click **Refresh** to update the status.

If your browser supports refresh, you can also use the URL `http://<yourhost>/server-status?refresh=5` to refresh every 5 seconds. As shown in Figure 16-19, you can see (along with additional information) the number of requests currently being processed, and the number of idle servers.

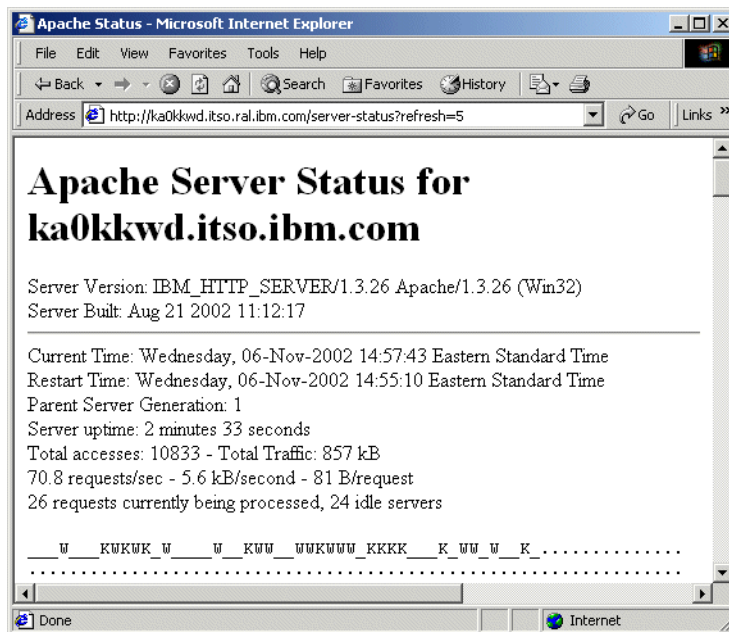


Figure 16-19 IBM HTTP Server status page

## 16.9 Log Analyzer

The Log Analyzer is a GUI tool that allows the user to view any logs generated with the loganalyzer TraceFormat, such as the IBM service log file and other traces using this format. It can take one or more service logs or trace logs, merge all the data, and display the entries in sequence.

Log Analyzer is shipped with an XML database, the *symptom database*, which contains entries for common problems, reasons for the errors, and recovery steps. The Log Analyzer compares every error record in the log file to the internal set of known problems in the symptom database and displays all the matches. This allows the user to get error message explanations and information such as why the error occurred and how to recover from it. Figure 16-20 shows the conceptual model of Log Analyzer.

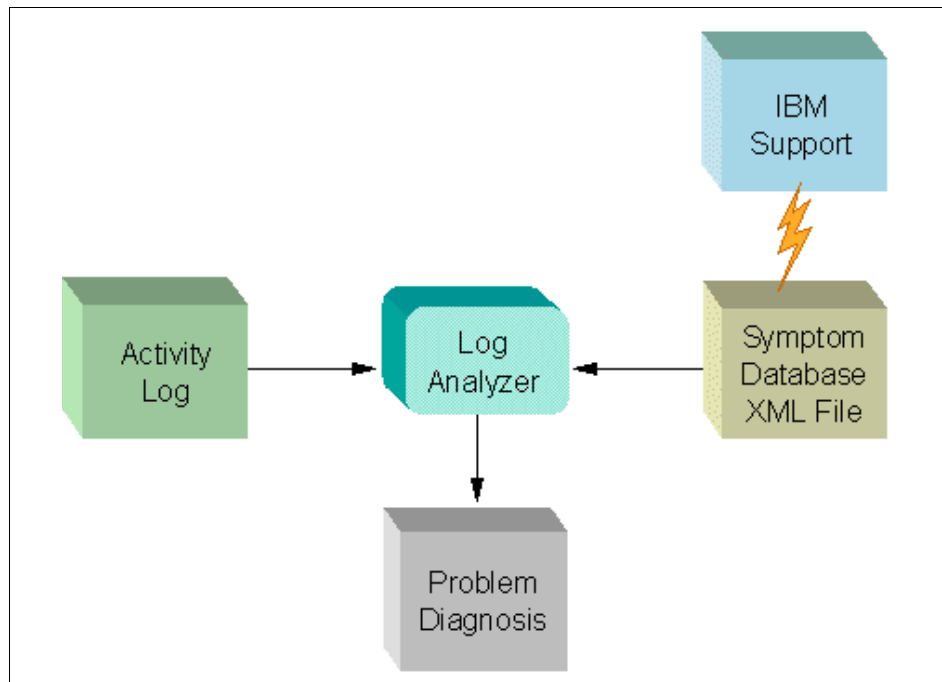


Figure 16-20 Conceptual model of Log Analyzer

For a detailed description about how to use Log Analyzer, refer to Section 18.5 of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195.

## 16.10 ThreadAnalyzer Technology preview

ThreadAnalyzer is an analysis tool that expedites problem determination for hangs and performance bottlenecks. The cause of hang conditions may be difficult to isolate through direct examination of application code, due to the interaction among concurrent threads. Using ThreadAnalyzer to view thread dumps from a running system provides a snapshot of what concurrent threads are doing at a moment in time, thereby illuminating areas of code that are expensive in terms of processing time. Or areas where threads are blocked and waiting for a resource to become available or even deadlocked. Once these areas of code have been isolated, they can be optimized to alleviate bottlenecks.

ThreadAnalyzer has the ability to capture thread dumps and provide quick summary views. Thread usage can be analyzed at several different levels, starting with a high-level graphical view, and drilling down to a detailed tally of individual threads. If any deadlocks exist in the thread dump, ThreadAnalyzer will detect and report them.

ThreadAnalyzer is supported on Solaris, AIX, Windows, HP, and Linux on Intel and IBM @server zSeries®.

Some typical cases where thread analysis is useful:

- ▶ Thread deadlock (hang conditions) detection
- ▶ Downstream response waits
- ▶ Upstream reply waits
- ▶ Low or no load reaching application server
- ▶ Large remote object size
- ▶ Thread synchronization

### 16.10.1 Download and installation of ThreadAnalyzer

The ThreadAnalyzer does not come with the WebSphere Application Server software. You can download it separately from:

<http://www.ibm.com/developerworks/websphere/downloads/techpreviews.html>

After downloading it, extract `ta.zip`. A directory called `threadanalyzer` will be created. This directory contains the code and commands needed for running ThreadAnalyzer.

### 16.10.2 Using ThreadAnalyzer

It might be hard for you to read the text-based Java stack trace file, which often contains hundreds or thousands of lines. Thread Analyzer will assist you in reading and analyzing the trace.

## Starting Thread Analyzer

To run Thread Analyzer, you first need to set the WAS\_HOME variable to the install path for WebSphere Application Server. Then start ThreadAnalyzer:

- ▶ On Windows platforms, run the **tagui.cmd** command from the threadanalyzer\bin directory. Example 16-5 shows how to start Thread Analyzer. Figure 16-21 shows the GUI when the ThreadAnalyzer is first started.
- ▶ For Unix platforms, run **chmod u+x tagui.sh**. Run the tagui.sh command from the threadanalyzer/bin directory.

### Example 16-5 Starting ThreadAnalyzer

```
C:\ThreadAnalyzer\bin>set WAS_HOME=c:\WebSphere\AppServer  
C:\ThreadAnalyzer\bin>tagui.cmd
```

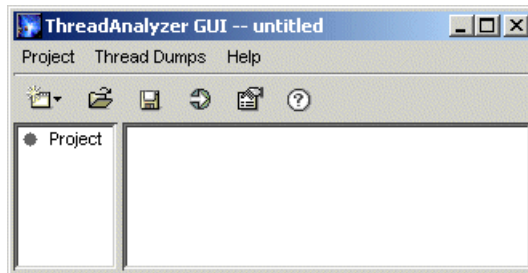


Figure 16-21 Starting Thread Analyzer

The ThreadAnalyzer GUI window consists of the menu bar at the top, the toolbar just below the menu bar, on the left-hand side you find the Project tree, and on the right-hand side the analysis result pane.

## ThreadAnalyzer project

A project is the top-level storage structure for ThreadAnalyzer as shown in Figure 16-22 on page 736.

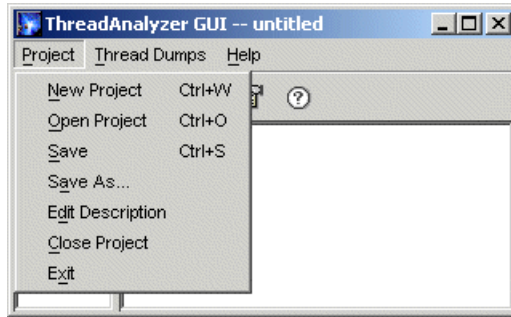


Figure 16-22 ThreadAnalyzer - Project menu

- ▶ **New Project:** A new project is a project initially opened without thread dumps and without a title.
- ▶ **Open Project:** Opening an existing project will open a previously saved project and display all of the thread dumps that were stored in that project.
- ▶ **Save:** The Save project option allows you to save the current project with the current project name.
- ▶ **Save As:** The Save As project option allows you to save the current project under a different project name.
- ▶ **Edit Description:** This allows you to add a description of the project.
- ▶ **Close Project:** The Close Project option closes the project from ThreadAnalyzer. If the project has not been saved, you will be prompted to save the project.

### ThreadAnalyzer thread dumps

Initially, a project has no thread dumps. A thread dump will initially be named TDx where x is the thread dump's ordered number in the project. You can rename a thread dump, remove a thread dump, edit a dump description, highlight, and Setup Options as shown in Figure 16-23 on page 737.



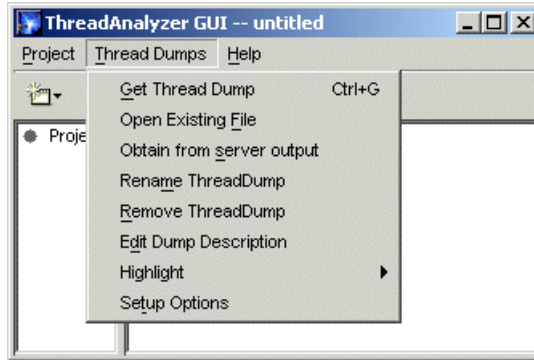


Figure 16-23 ThreadAnalyzer - Thread Dumps menu

## Obtaining a thread dump to analyze

There are several ways to get a Java stack trace (or thread dump) into the Thread Analyzer.

### ***From an existing Java stack trace file***

1. Select **ThreadDumps -> Open Existing File** as shown in Figure 16-23.
2. Select and open the Java stack trace file (normally named `javacore*.txt`) you wish to analyze. The shortcut to opening an existing file is **CTRL-O**.

### ***From a server output log file***

You can take a Java stack trace source from the application server's log files, `<stderr>` or `<stdout>` files, or any other snippets of files that contain a thread dump as shown in Figure 16-24 on page 738.

1. Select **ThreadDumps -> Obtain from server output**.
2. Click **Add file(s)** on the new pop-up window to select one or more files.
3. Click the **Process** button to extract the Java stack trace parts from the files.

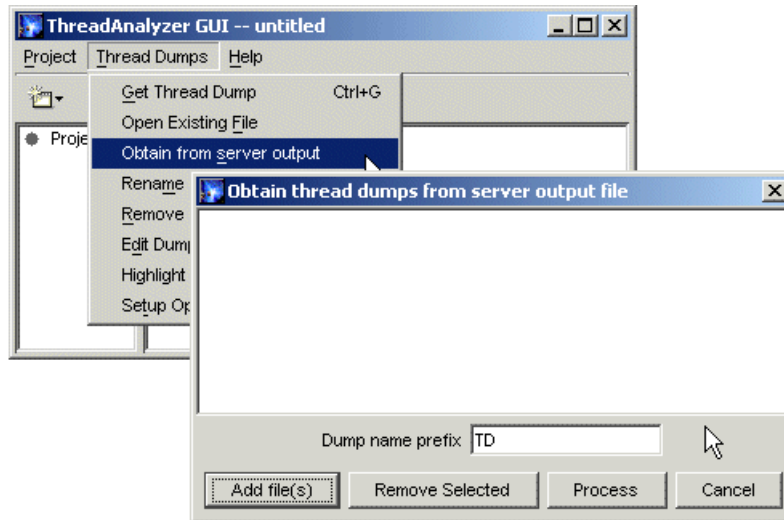


Figure 16-24 ThreadAnalyzer - Obtain from server output

### **Obtaining a thread dump in real time**

You can also generate a real-time thread dump while the application server is running:

1. Before getting a thread dump from an application server, you must first define the Setup Options. Select **ThreadDumps -> Setup Options**, as shown in Figure 16-25 on page 739.
  - a. In the pop-up window, enter the WebSphere Application Server installation directory if it is not the one displayed as “Current WAS Home”.
  - b. Type in the name of the application server you want to analyze if it is not the Current Application Server Name.
  - c. Verify or select the SOAP port number. You should first check this port number from the WebSphere Administrative Console by selecting **Servers -> Application Servers** in the console navigation tree. Open the server configuration and select **End Points** from the Additional Properties pane. The port number is found in the **SOAP\_CONNECTOR\_ADDRESS** configuration.
  - d. Change or take the default for the wait time, stack trace saving options, and log options.
  - e. Now, you can generate a thread dump from a running application server by clicking the **OK** button.

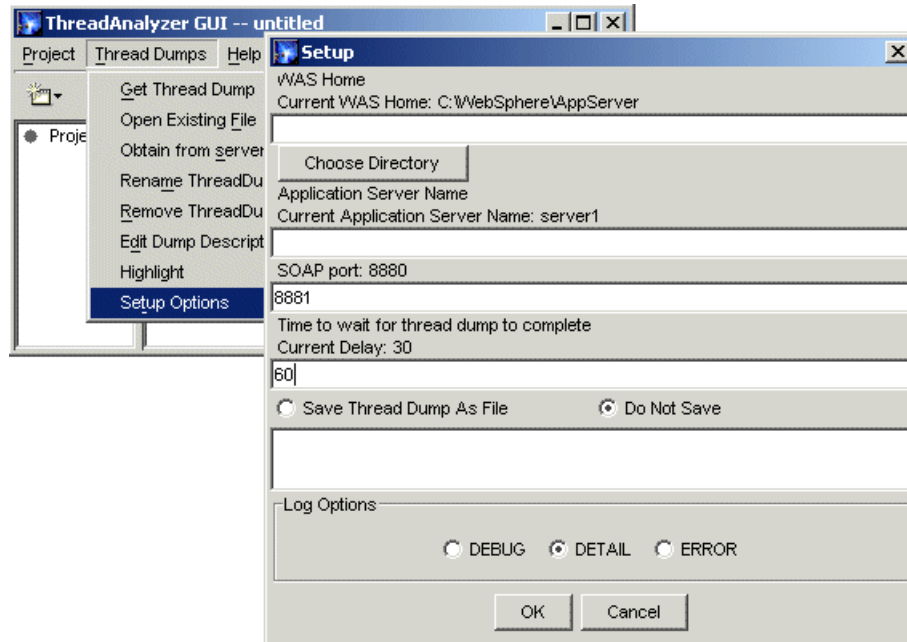


Figure 16-25 ThreadAnalyzer - Setup Options

2. After you have defined the Setup Options, click **ThreadDumps** -> **Get ThreadDump** as shown in Figure 16-26 or simply press **Ctrl+G**.

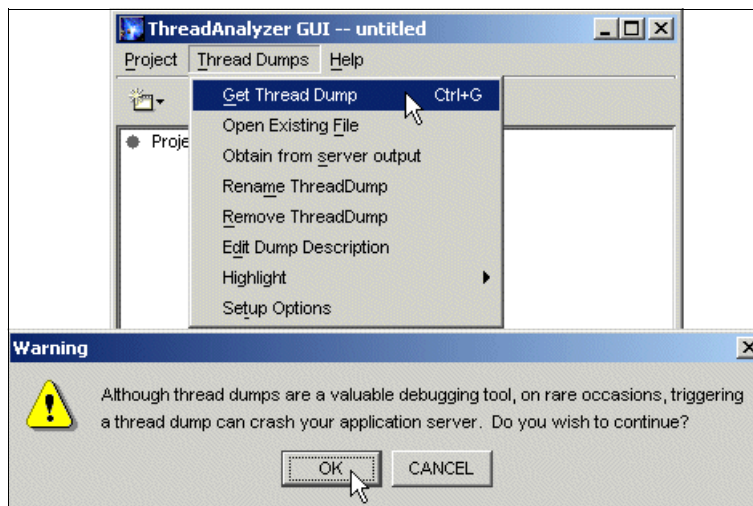


Figure 16-26 ThreadAnalyzer - Getting ThreadDump

## Navigating in ThreadAnalyzer

If you have successfully generated or loading the Java stack trace or thread dump you should see something like in Figure 16-27.

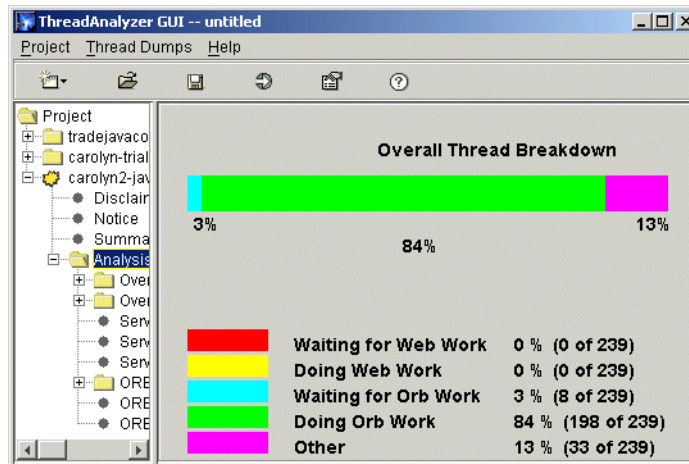


Figure 16-27 ThreadAnalyzer -- Result of ThreadDump

There are several views available that you can use for your analysis.

- **Overall thread analysis:** This view is the overall thread analysis. If threads exist in the thread dump, they will be listed here.
- **Overall monitor analysis:** This view shows the overall monitor analysis and Interpreting Deadlock Results as shown in Figure 16-29 on page 741. Expand this tree node to see a list of all monitors in this thread dump.
- You can see that there are no servlet engine worker threads waiting or doing anything. However, you will note that the ORB has eight threads waiting for work and 198 threads doing work. For more detailed information about what the threads are doing, you can use the overall thread analysis for all threads state in the left navigation tree, as shown in the figure as shown in Figure 16-28 on page 741. With this picture, you would have a strong supposition of a problem occurring in the ORB.

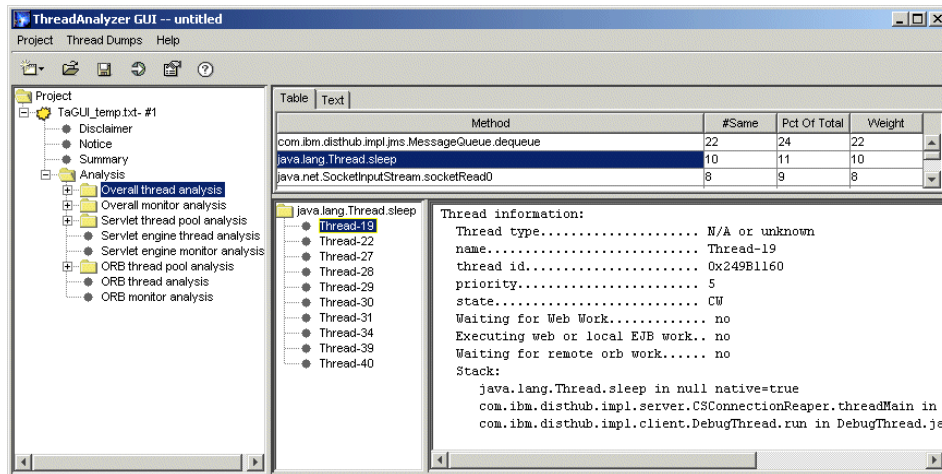


Figure 16-28 ThreadAnalyzer - Overall thread analysis

### 16.10.3 Automatic deadlock detection

Thread deadlocks can be very hard to debug. To help you in this, the ThreadAnalyzer provides information on threads holding locks and threads waiting on locks. The Deadlock Detection Report is displayed when one or more deadlocks are detected as shown in Figure 16-29.

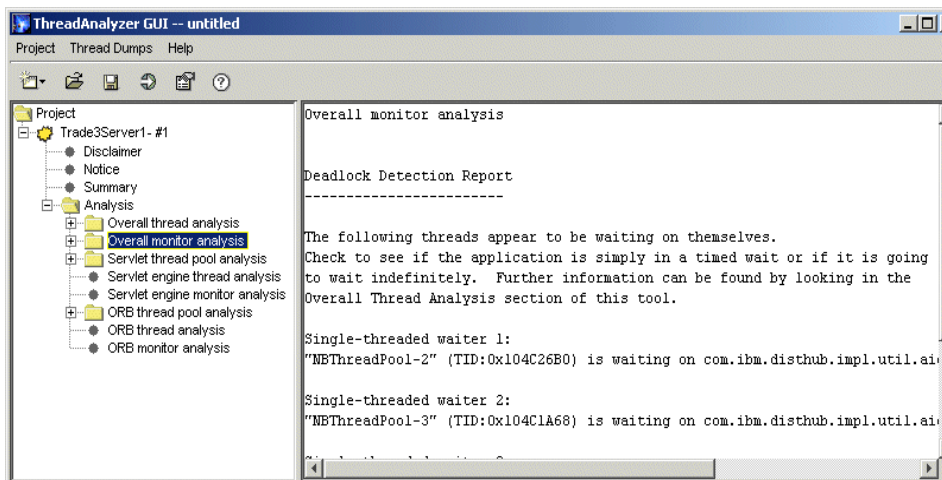


Figure 16-29 ThreadAnalyzer - Overall Monitor Analysis and Deadlock Detection Report

Deadlock information is displayed differently between AIX, Windows, Linux, zLinux, HP, and Solaris. The output format for AIX, Windows, Linux, and zLinux is shown in Example 16-6.

*Example 16-6 Deadlock Detection Report: Output format for AIX, Windows, Linux, and zLinux*

---

Overall monitor analysis

Deadlock Detection Report

-----

The following threads appear to be in a circular deadlock.  
Further information can be found by looking in the Overall Thread Analysis section of this tool.

Multi-threaded deadlock 1:

```
"inst: 0 - thd:0" of (sys:0x136AFC68) (TID:0x4D3F3E0)
    Holding Resource: java.lang.Object@4D517F0/4D517F8
    Thread Waiting: "inst: 0 - thd:4" (sys:0x136B1030) (TID:0x4D3F2A0)
```

---

- ▶ The first line shows the thread named inst: 0 - thd:0 with thread identification numbers sys:0x136AFC68 and TID: 0x4D3F3E0.
- ▶ The next line shows that the thread inst: 0 - thd:0 is holding the resource called java.lang.Object@4D517F0/4D517F8.
- ▶ The following lines show that thread inst: 0 - thd:4 with thread identification numbers sys:0x136B1030 and TID:0x4D3F2A0 is waiting for the resource java.lang.Object@4D517F0/4D517F8.

The output format for HP and Solaris is shown in Example 16-7.

*Example 16-7 Deadlock Detection Report: Output format for HP and Solaris*

---

FOUND A JAVA LEVEL DEADLOCK:

-----

```
"Servlet.Engine.Transports : 3":
    waiting to lock monitor 0xbc360 (object 0xea3383c8, a java.lang.Object),
    which is locked by "Servlet.Engine.Transports : 0"
```

---

- ▶ The first line shows that the thread Servlet.Engine.Transports : 3 is waiting for the thread Servlet.Engine.Transports : 0 to release the lock on the monitor (object 0xea3383c8, a java.lang.Object).
- ▶ In order to further analyze the cause of the deadlock, look for the thread's stack trace in the Overall Thread Analysis. The methods in the stack trace can be correlated to the source code in most cases.

## 16.10.4 ThreadAnalyzer usage example

This section explains how you can use ThreadAnalyzer to determine the reason for a performance problem. The example assumes a Web application is running with the file serving servlet enabled. For each request to the Web container, a 100 KB file is served. The expected transaction rate is about 10 requests/second. However, the observed rate is only about two requests/second. Why is the performance so bad? How can you diagnose this problem?

Use ThreadAnalyzer to capture several successive thread dumps from the JVM over a short time period, in order to observe any patterns in thread activity. Remember, the thread dumps are captured using the Ctrl+G keyboard shortcut or the menu option Get Thread Dump. Each dump is then parsed and sorted into views that are accessible through the expandable tree in the left-hand pane. The collection of dumps can also be saved in ThreadAnalyzer projects for later review.

After capturing a thread dump, ThreadAnalyzer's analysis graph shows all of the work being done by the servlet engine as shown in Figure 16-30.

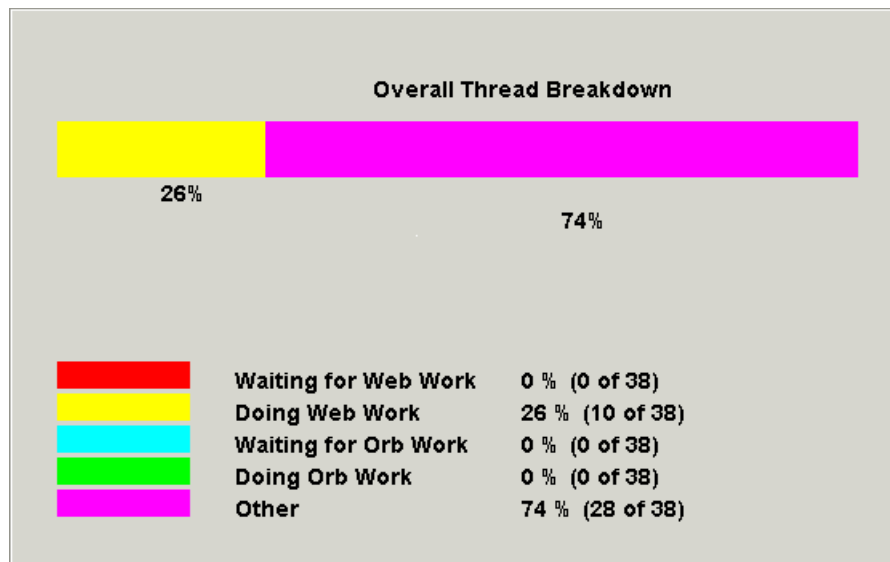


Figure 16-30 ThreadAnalyzer's analysis graph

The 26% of the graph represents the servlet threads that are responding to requests. All servlet threads appear to be busy, yet how can you reconcile this with the observed transaction rate of only two requests/second? You need to know what method(s) these threads were executing at the time the dump was

captured. To investigate further, look at the Servlet thread pool analysis view as detailed in Figure 16-31.

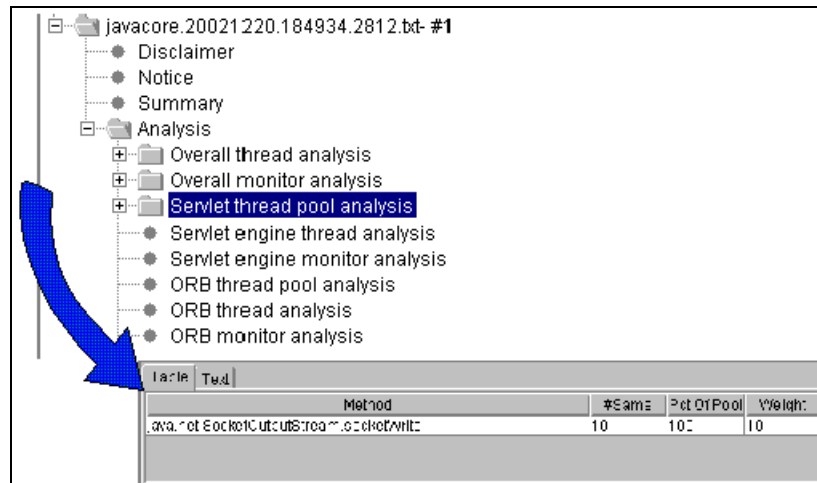


Figure 16-31 Servlet thread pool analysis view

This view shows 100% of servlet threads in the socketWrite call. Expanding the tree node reveals the list of servlet threads. Selecting a thread displays its stack in the right-hand pane as shown in Example 16-8.

#### Example 16-8 Thread stack

```
java.net.SocketOutputStream.socketWrite
  java.net.SocketOutputStream.write
    com.ibm.ws.io.Stream.write
      com.ibm.ws.io.WriteStream.flushMyBuf
        com.ibm.ws.io.WriteStream.write
          com.ibm.ws.http.ResponseStream.writeChunk
            com.ibm.ws.http.ResponseStream.write
              com.ibm.ws.io.WriteStream.flushMyBuf
                com.ibm.ws.io.WriteStream.write
                  com.ibm.ws.webcontainer.http.HttpConnection.write
                    com.ibm.ws.webcontainer.srp.SRPCConnection.write
                      com.ibm.ws.webcontainer.srt.SRTOutputStream.write
                        com.ibm.ws.webcontainer.srt.BufferedServletOutputStream.writeOut
                          com.ibm.ws.webcontainer.srt.BufferedServletOutputStream.write
                            com.ibm.ws.webcontainer.servlet.MyMainServlet.doGet
                              javax.servlet.http.HttpServlet.service
```

Read the stack from bottom to top to see that an instance of HttpServlet named MyMainServlet has been invoked. In its doGet method, the servlet performs a write operation to a BufferedServletOutputStream. The remaining calls up the



stack are all internal details of opening an `HttpConnection`. The top of the stack shows the thread in the `socketWrite` call at the time the dump was captured.

The content of what is being written to the socket is not as important as the observation that the servlet is sending a response back to the client. In isolation, one thread caught in a `socketWrite` would not be any cause for concern. However ThreadAnalyzer's summary view reports that 100% of the servlet threads are in this same call. Examining consecutive thread dumps reveals a similar pattern of activity. What does this tell you?

This is unusual, since under normal activity you would expect to see threads in a variety of method calls. Seeing so many threads in the `socketWrite` method consistently over a short time period indicates that the `socketWrite` call is requiring a longer time period to complete, relative to other method calls. You can now rule out the application server as the source of the bottleneck, and focus on the network as the source of the problem. After assigning a network engineer to examine the network path from the application server to the client, it turns out that the server's network adapter had been inadvertently set to half-duplex instead of full. Setting the network adapter to full-duplex cleared the bottleneck.

In summary, ThreadAnalyzer facilitates easier problem determination through the use of thread dumps. By quickly pinpointing areas of thread contention, problem turnaround times can be improved, resulting in reduced support costs as well as overall increased customer satisfaction.

### 16.10.5 Getting online help

ThreadAnalyzer has HTML-based online help. To see the help pages, select **Help -> Help Topics** from the main menu.

Additional information can also be found in 18.6, "ThreadAnalyzer technology preview" of the redbook *IBM WebSphere Version 5.0 System Management and Configuration*, SG24-6195.

## 16.11 Performance Advisors

When it comes to understanding how to tune WebSphere applications, it often feels like we are running with our headlights off! With IBM WebSphere Application Server, V5.0.2 and V5.1, the Performance Advisors come to the rescue. There are two performance advisors that provide suggestions to help tune systems for optimal performance. Both advisors use the Performance Monitoring Infrastructure (PMI) data to suggest configuration changes to the WebSphere thread pools, connection pools, prepared statement cache, session cache, heap size, etcetera.

The two performance advisors are:

- ▶ Runtime Performance Advisor
- ▶ Performance Advisor in Tivoli Performance Viewer (TPV Advisor)

The first advisor, the Runtime Performance Advisor, which executes in the application server process, is configured through the WebSphere Administrative Console. Running in the application server's JVM, this advisor periodically checks for inefficient settings, and issues recommendations as standard WebSphere warning messages. These recommendations are displayed both as warnings in the Administrative Console under WebSphere Runtime Messages in the WebSphere Status panel and as text in the SystemOut.log file. Enabling the Runtime Performance Advisor has minimal system performance impact.

The second, the TPV Advisor, runs in the Tivoli Performance Viewer (TPV) and provides recommendations on inefficient settings at a specific point in time. The TPV Advisor provides more extensive advice than the Runtime Performance Advisor. For example, TPV provides advice on setting the dynamic cache size, setting the JVM heap size, and using the DB2 Performance Configuration Wizard.

Table 16-6 gives a summary of the two Performance Advisors.

*Table 16-6 Performance advisors summary*

	<b>Runtime Performance Advisor</b>	<b>Performance Advisor in Tivoli Performance Viewer (TPV)</b>
Location of execution	Application server	TPV client
Location of tool	Administrative console	TPV
Output	SystemOut.log file and WebSphere run-time messages in WebSphere status panel in the Administrative Console	TPV graphical user interface (GUI)
Frequency of operation	Every 10 seconds in background	When you select to refresh in TPV

	Runtime Performance Advisor	Performance Advisor in Tivoli Performance Viewer (TPV)
Types of advice	<ul style="list-style-type: none"> <li>▶ ORB service thread pools</li> <li>▶ Web container thread pools</li> <li>▶ Connection pool size</li> <li>▶ Persisted session size and time</li> <li>▶ Prepared statement cache size</li> <li>▶ Session cache size</li> </ul>	<ul style="list-style-type: none"> <li>▶ ORB service thread pools</li> <li>▶ Web container thread pools</li> <li>▶ Connection pool size</li> <li>▶ Persisted session size and time</li> <li>▶ Prepared statement cache size</li> <li>▶ Session cache size</li> <li>▶ Dynamic cache size</li> <li>▶ JVM heap size</li> <li>▶ DB2 Performance Configuration Wizard</li> </ul>

Figure 16-32 shows the simplified architecture of the Performance Advisors.

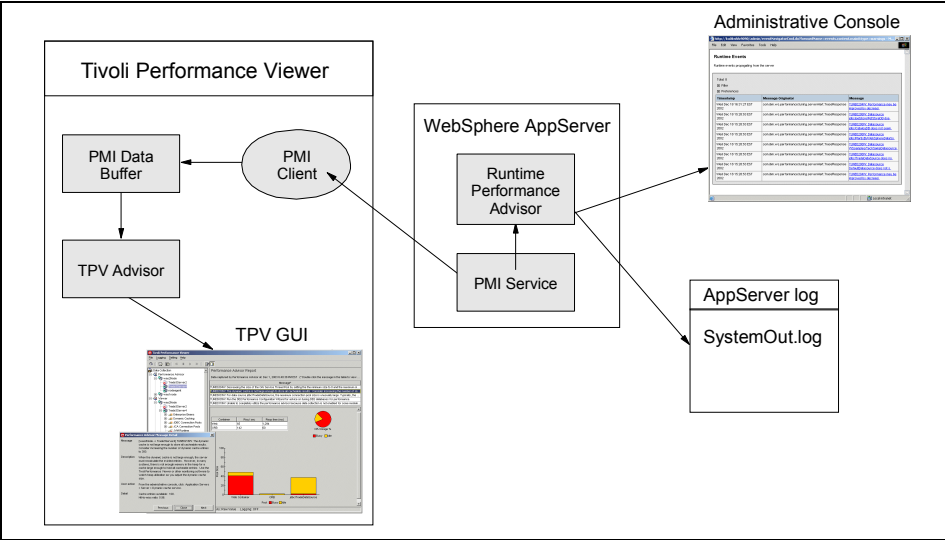


Figure 16-32 Simplified performance advisors architecture

For more information about PMI and TPV, refer to 16.2, “Performance Monitoring Infrastructure” on page 687 and 16.3, “Using Tivoli Performance Viewer” on page 705.

### 16.11.1 Runtime Performance Advisor configuration settings

This section lists the various settings within the Administrative Console under the Runtime Performance Advisor Configuration panel.

To view the Runtime Performance Advisor configuration page, click **Servers -> Application Servers -> <server\_name> -> Runtime Performance Advisor Configuration**.

### ***Configuration tab***

- ▶ **Enable Runtime Performance Advisor**  
Specifies whether the Runtime Performance Advisor runs.
- ▶ **Calculation Interval**  
PMI data is taken over an interval of time and averaged to provide advice. The interval specifies the length of the time over which data is taken for this advice. Therefore, details within the advice messages will appear as averages over this interval.
- ▶ **Maximum warning sequence**  
The maximum warning sequence refers to the number of consecutive warnings issued before the threshold is updated. For example, if the maximum warning sequence is set to 3, then the advisor only sends three warnings to indicate that the prepared statement cache is overflowing. After that, a new alert is only issued if the rate of discards exceeds the new threshold setting.
- ▶ **Number of processors**  
Specifies the number of processors on the server.

### ***Runtime tab***

The four configuration settings from the Configuration tab are also available on the Runtime tab.

In addition there is a Restart button on this tab. Selecting Restart reinitializes the Runtime Performance Advisor with the last information saved to disk.

Note that this action also resets the state of the Runtime Performance Advisor. For example, the current warning count is reset to zero for each message.

## **16.11.2 Advice configuration settings**

WebSphere Application Server also allows you to enable and disable advice in the Advice Configuration panel. Some advice applies only to certain configurations, and can only be enabled for those configurations. For example, Unbounded ORB Service Thread Pool Advice is only relevant when the ORB Service thread pool is unbounded, and can only be enabled when the ORB thread pool is unbounded. Below are the various settings within the Administrative Console under the Advice Configuration Settings panel:

To view this administrative page, click **Servers -> Application Servers -> <server\_name> -> Runtime Performance Advisor Configuration -> Advice Configuration**.

### ***Configuration tab***

- ▶ Advice name  
Specifies the advice that you can enable or disable.
- ▶ Advice applied to component  
Specifies the WebSphere Application Server component to which the runtime performance advice applies.
- ▶ Advice status  
Specifies whether advice is stopped or started.  
  
There are only two values - Started and Stopped. Started means that the advice runs if the advice applies. Stopped means that the advice does not run.

### ***Runtime tab***

- ▶ Advice name and Advice applied to component are identical to the Configuration tab.
- ▶ Advice status  
  
On the Runtime tab, the advice status has one of three values - Started, Stopped or Unavailable. Started means that the advice is being applied. Stopped means that the advice is not applied. Unavailable means that the advice does not apply to the current configuration (such as Persisted Session Size advice in a configuration without persistent sessions).

## **16.11.3 Using the Runtime Performance Advisor**

In order to obtain advice, you must first enable the performance monitoring service through the Administrative Console and restart the server. If running Network Deployment, you must enable the PMI service on both the server and on the Node Agent and restart the server and Node Agent.

The Runtime Performance Advisor enables the appropriate monitoring counter levels for all enabled advice. If there are specific counters that are not wanted, disable the corresponding advice in the Runtime Performance Advisor panel, and disable unwanted counters. See “Enabling the PMI service” on page 697 for details. Then do the following:

1. In the WebSphere Administrative Console select **Servers -> Application Servers** from the navigation tree.

2. Click **<server\_name> -> Runtime Performance Advisor Configuration**.
3. On the **Configuration** tab select the **Number of Processors**.  
Selecting the appropriate settings for the system's configuration ensures accurate performance advice.
4. Optionally, select the **Calculation Interval**.
5. Again optionally, select the **Maximum warning sequence**.
6. Click **Apply** and **Save** your changes.
7. Select the **Runtime** tab and click **Restart**.
8. Simulate a production level load.

For load testing tools and how to perform the load testing, see 19.2, “Tools of the trade” on page 823.

If you are using the Runtime Performance Advisor in a test environment, or doing any other tuning for performance, simulate a realistic production load for your application. The application should run this load without errors. This simulation includes the number of concurrent users typical for peak periods, and drives system resources, such as CPU and memory to the levels expected in production. The Runtime Performance Advisor only provides advice when CPU utilization exceeds a sufficiently high level.

9. Once a stable production level load is reached, select the check box to **Enable** the Runtime Performance Advisor. This way, you will achieve the best results for performance tuning. Click **OK**.
10. Select **Warnings** in the Administrative Console under the WebSphere Runtime Messages in the WebSphere Status panel or look at the SystemOut.log file, located in the <install\_root\logs\server\_name> directory to view tuning advice. Some messages are not issued immediately.
11. Update the product configuration for improved performance, based on the received advice.

**Important:** As with any analysis and tuning, make sure that only one parameter is changed, and then the results monitored. This provides an effective method of backing out of the change if it proves detrimental to the environment. Also, making multiple changes could result in undesired results, because many options are dependent on other settings.

Although the performance advisors attempt to distinguish between loaded and idle conditions, misleading advice might be issued if the advisor is enabled while the system is ramping up or down. This result is especially likely when running short tests. Although the advice helps in most configurations, there might be situations where the advice hinders

performance. Due to these conditions, advice is not guaranteed. Therefore, test the environment with the updated configuration to ensure it functions and performs as expected.

**Tips for using the Runtime Performance Advisor:**

- ▶ Enable Performance Monitoring Service in the application server *and* in the Node Agent if running Network Deployment.
- ▶ Be sure to configure the correct number of processors.
- ▶ Simulate a production level load.
  - Ensure the application runs without exceptions/errors.
  - Runtime Performance Advisor only provides advice when CPU utilization is high.
- ▶ Once production load level is reached, enable the Runtime Performance Advisor.
- ▶ Apply advice, restart the application server, and re-test.
- ▶ More details can be found in the WebSphere InfoCenter section “Using the Runtime Performance Advisor”.

## 16.11.4 Runtime Performance Advisor output

After completing the configuration steps, the Advisor will then begin to report recommendations into the SystemOut.log. Example 16-9 shows sample output from the Advisor.

*Example 16-9 Sample output from the Runtime Advisor*

---

```
[6/11/03 15:20:42:484 EDT] 6a1d6f88 TraceResponse W TUNE0208W: Data source
jdbc/TradeDataSource does not seem to be in use. If the data source is used
occasionally, then decrease the number of connections in the pool, by setting
minConnections to 0, and maxConnections to 3. If the data source is never used,
then delete the data source.
Additional explanatory data follows.
Pool utilization: 0%.
This alert has been issued 1 time(s) in a row. The threshold will be updated to
reduce the overhead of the analysis.
[6/11/03 15:24:54:953 EDT] 6a1d6f88 TraceResponse W TUNE0214W: The session
cache for Trade3#trade3Web.war is smaller than the average number of live
sessions. Increasing the session cache size to at least 1,300 may improve
performance.
Additional explanatory data follows.
Session cache size (the maximum in memory session count): 1,000.
Current live sessions: 1,300.
Average live sessions over the last sampling interval: 1,300.
```

This alert has been issued 1 time(s) in a row. The threshold will be updated to reduce the overhead of the analysis.

In addition, the Runtime Advisor recommendations can also be displayed in the Administrative Console through the WebSphere Runtime Messages. The Runtime Advisor messages are displayed as warnings. Figure 16-33 shows a sample of the output from the advisor.

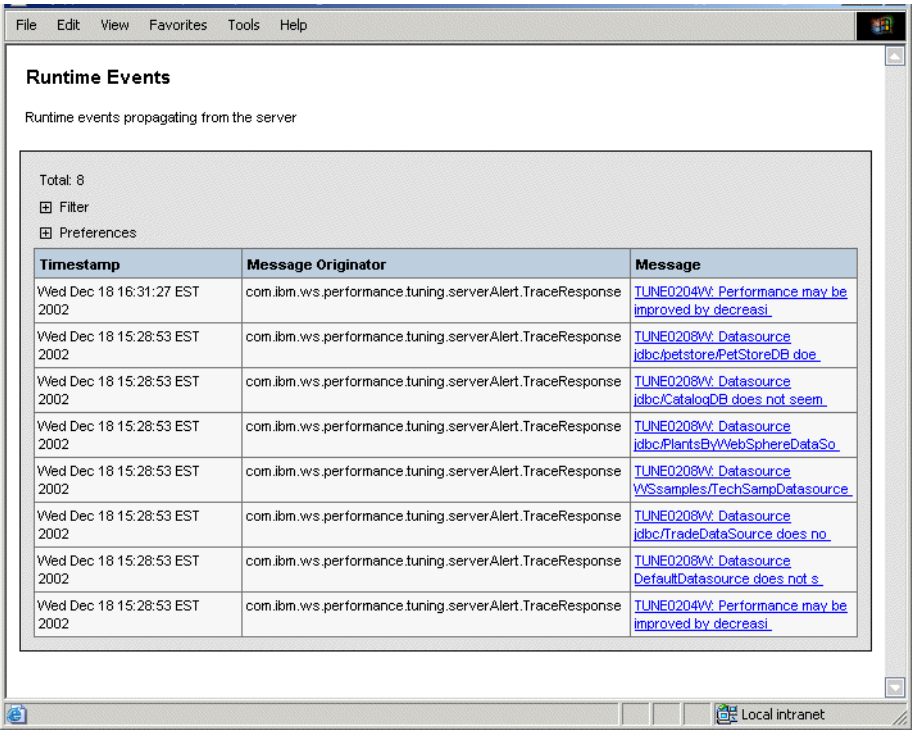


Figure 16-33 Runtime Events display of Runtime Advisor output

By following the link for the detailed message, Figure 16-34 on page 753 is an example of such message displayed.



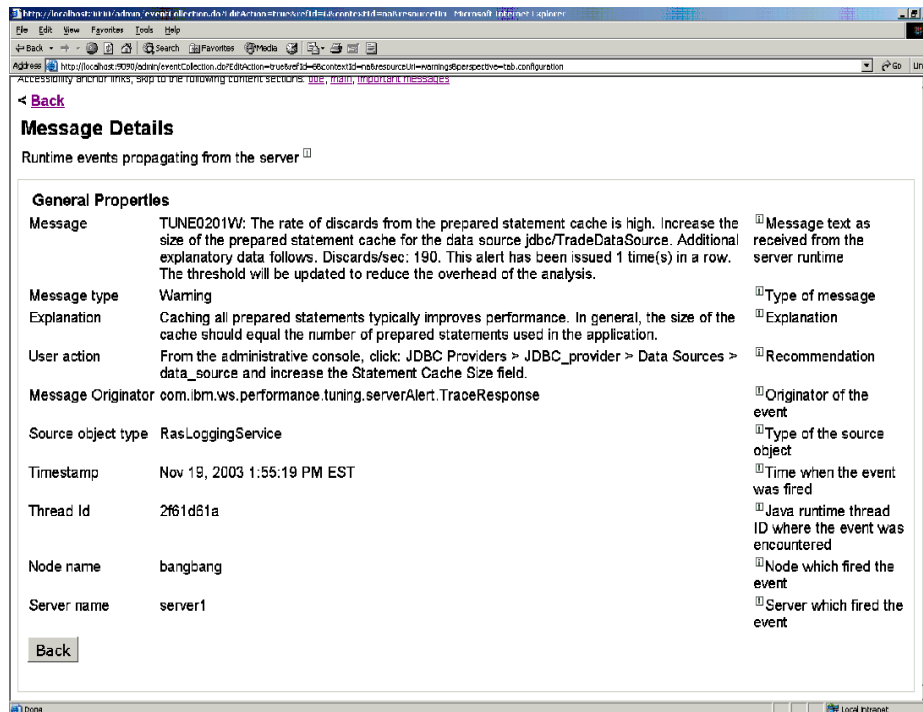


Figure 16-34 Detail of Runtime Advisor message

As mentioned in “Using the Runtime Performance Advisor” on page 749, use these recommendations as input to change your configuration, restart the application server(s), and re-test.

### 16.11.5 TPV Advisor configuration

The TPV Advisor is part of the TPV, therefore the connection and parameters settings explained in “Running Tivoli Performance Viewer” on page 708 apply.

Once the connection is established, the application will display a warning about any application servers that are defined, but have not had the PMI service enabled. In most cases, it would be intentional that the application server does not have monitoring enabled. The Performance Advisor then displays a window similar to Figure 16-12 on page 713 with **Performance Advisor** under the **Data Collection** tree, it shows in Figure 16-35 on page 758. Specifics of the application and the settings can be reviewed in 16.3, “Using Tivoli Performance Viewer” on page 705.

## 16.11.6 Using TPV Advisor

The Performance Advisor in Tivoli Performance Viewer (TPV) provides advice to help tune systems for optimal performance and gives recommendations on inefficient settings by using collected Performance Monitoring Infrastructure (PMI) data. Advice is obtained by selecting the Performance Advisor icon in TPV. The Performance Advisor in TPV provides more extensive advice than the Runtime Performance Advisor. For example, TPV provides advice on setting the dynamic cache size, setting the JVM heap size and using the DB2 Performance Configuration Wizard.

In order to obtain advice, you must first enable the performance monitoring service as shown in “Enabling the PMI service” on page 697, then do the following steps:

1. Enable data collection.

The monitoring levels that determine which data counters are enabled can be set dynamically, without restarting the server. These monitoring levels and the data selected determine the type of advice you obtain. The Performance Advisor in TPV uses the standard monitoring level; however, the Performance Advisor in TPV can use a few of the more expensive counters (to provide additional advice) and provide advice on which counters can be enabled. This action can be completed in one of the following ways:

- a. Enable data collection through the Administrative Console.
  - b. Enable performance monitoring services through Tivoli Performance Viewer.
  - c. Enable performance monitoring services using the command line.
2. Start the Tivoli Performance Viewer. This can be done in two ways:
    - a. Start performance monitoring from the command line. See Example 16-10.

Go to the <install\_root>/bin directory and run the **tperfviewer** script.

You can specify the host and port in Windows NT and 2000 environments as follows:

```
tperfviewer.bat <host_name> <port_number> <connector_type>
```

**Note:** On the AIX and other UNIX platforms, use:

```
tperfviewer.sh <host_name> <port_number> <connector_type>
```

*Example 16-10 Start performance monitoring from the command line*

---

```
tperfviewer.bat localhost 8879 SOAP
```

Connector\_type can be either SOAP or RMI.

8879 is the default ND port for SOAP connector.  
9809 is the default ND port for RMI connector

---

- b. Click **Start -> Programs -> IBM WebSphere -> Application Server v5.1 -> Tivoli Performance Viewer.**

Tivoli Performance Viewer detects which package of WebSphere Application Server you are using and connects using the default Remote Method Invocation (RMI) connector port. If the connection fails, a dialog is displayed to provide new connection parameters.

You can connect to a remote host or a different port number, by using the command line to start the performance viewer.

3. Simulate a production level load. See step 8 on page 750 for details.
4. Optionally store data in a log file:
  - a. Click **Logging -> On** or click the **Logging** icon.
  - b. Specify the name, location, and format type of the log file in the **Save** dialog box.

The **Files of type** field allows an extension of \*.perf for binary files or \*.xml for XML format. The XML format provides the flexibility that enables analysis by using third-party tools.

**Note:** The \*.perf files may not be compatible between fix levels.

- c. Click **OK**.
5. You could also replay a performance data log file:
  - a. Click **Data Collection** in the navigation tree.
  - b. Click the **Log** radio button in the **Performance data** from field.
  - c. Click **Browse** to locate the file that you want to replay or type the file pathname in the **Log** field.
  - d. Click **Apply**.
  - e. Play the log by using the **Play** icon or click **Setting -> Log Replay -> Play**.
6. Refresh data as needed.
  - a. Select one or more resources in the Resource Selection panel.
  - b. Click **File -> Refresh**. Alternatively, click the **Refresh** icon or right-click the resource and select **Refresh**.

Clicking refresh with a server selected under the viewer icon causes TPV to

- Query the server for new PMI and product configuration information.

Clicking refresh with a server selected under the advisor icon causes TPV to:

- Refresh advice that is provided in a single instant in time.
  - Not query the server for new PMI and product configuration information.
7. Tuning advice appears when the Advisor icon is chosen in the TPV Performance Advisor. Double-click an individual message for details. Since PMI data is taken over an interval of time and averaged to provide advice, details within the advice message appear as averages.
  8. Since Tivoli Performance Viewer refreshes advice at a single instant in time, take the advice from the peak load time.
  9. Update the product configuration for improved performance, based on the advice.

**Important:** As with any analysis and tuning, make sure that only one parameter is changed, and then the results monitored. This provides an effective method of backing out of the change if it proves detrimental to the environment. Also, making multiple changes could result in undesired results, because many options are dependent on other settings.

Although the performance advisors attempt to distinguish between loaded and idle conditions, misleading advice might be issued if the advisor is enabled while the system is ramping up or down. This result is especially likely when running short tests. Although the advice helps in most configurations, there might be situations where the advice hinders performance. Due to these conditions, advice is not guaranteed. Therefore, test the environment with the updated configuration to ensure it functions and performs well.

10. Optionally you can clear values from tables and charts.
  - a. Click one or more resources in the Resource Selection panel.
  - b. Click **Setting -> Clear Values**. Alternatively, right-click the resource and select **Clear Values**.
11. Another optional task is to reset the counters to zero.
  - a. Click one or more resources in the Resource Selection panel.
  - b. Click **Setting -> Reset to Zero**. Alternatively, right-click the resource and click **Reset to Zero**.

**Tips for using the TPV Advisor:**

- ▶ Enable Performance Monitoring Service in the application server *and* in the Node Agent if running Network Deployment.
- ▶ Set the monitoring level to standard.
  - Some JVM rules require monitoring level MAX and JVMPI.
- ▶ Start Tivoli Performance Viewer (tperfviewer.bat).
- ▶ Simulate production level load.
  - Ensure the application runs without exceptions/errors.
- ▶ Apply advice, restart the application server and re-test.
- ▶ More details can be found in the InfoCenter section “Using the Performance Advisor in Tivoli Performance Viewer”.

### 16.11.7 TPV Advisor output

To review the advisor suggestions, expand the Performance Advisor tree, expand the node that is executing the application server, and highlight the desired application server. Tivoli Performance Viewer polls and retrieves performance data. The TPV Advisor works in conjunction with Tivoli Performance Viewer using the TPV infrastructure. TPV Advisor provides advice based on analysis of the gathered data. When using the Tivoli Performance Viewer Performance Advisor in a production environment, be sure to obtain and use advice at your peak load. In a pre-production environment, simulate a production load and obtain and use advice at the peak load (see 19.2, “Tools of the trade” on page 823 for more information).

As the TPV Advisor executes, output is displayed in the Tivoli Performance Viewer application window. To see the details of a specific advisor message, double-click the message and a new pop-up window displays the exact details of the message which includes message, description, user action, details. Figure 16-35 on page 758 shows a sample of this output.

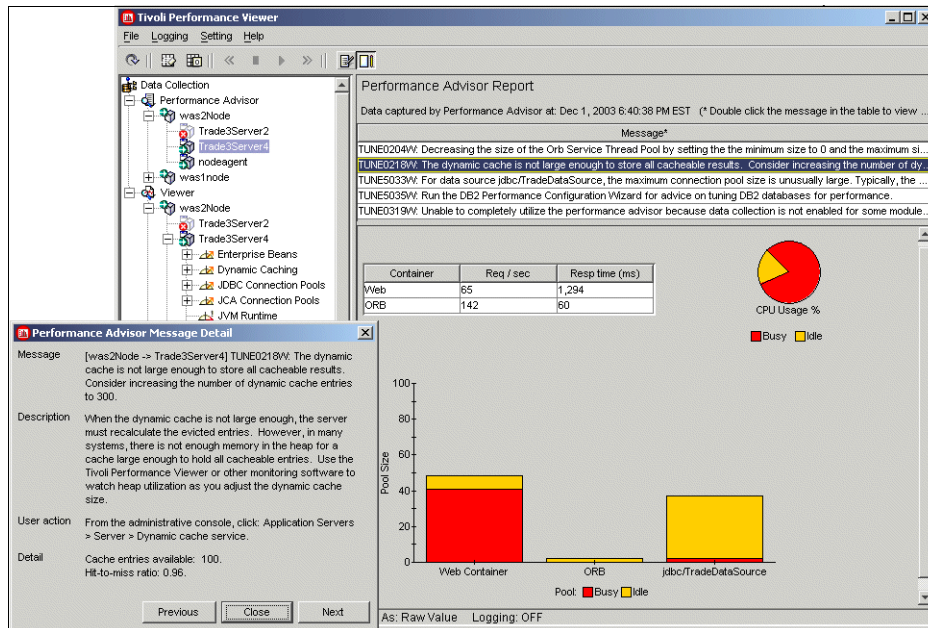


Figure 16-35 TPV Advisor output

## 16.12 Reference

For more information on the topics covered in this chapter, refer to:

- ▶ e-Pro mag.com - Ruth Willenborg: Monitoring Performance with WebSphere:  
<http://www.e-promag.com/eparchive//index.cfm?fuseaction=viewarticle&ContentID=1492&publicationid=13&PageView=Search&channel=2>
- ▶ IBM WebSphere Developer Technical Journal: Writing PMI applications using the JMX interface:  
[http://www.ibm.com/developerworks/websphere/techjournal/0402\\_qiao/0402\\_qiao.html](http://www.ibm.com/developerworks/websphere/techjournal/0402_qiao/0402_qiao.html)
- ▶ IBM WebSphere Developer Technical Journal: Writing a Performance Monitoring Tool Using WebSphere Application Server's Performance Monitoring Infrastructure API:  
[http://www.ibm.com/developerworks/websphere/techjournal/0202\\_rangaswamy/rangaswamy.html](http://www.ibm.com/developerworks/websphere/techjournal/0202_rangaswamy/rangaswamy.html)
- ▶ WebSphere Performance Diagnostics - Going beyond the Metrics:  
<http://www.sys-con.com/websphere/article.cfm?id=207>



## Development-side performance and analysis tools

This chapter discusses tools for development-side measuring and analyzing performance of WebSphere applications.

In particular we discuss the following:

- ▶ The profiling tools incorporated into IBM WebSphere Studio Application Developer V5.1.1 - called the Profiler throughout this chapter.
- ▶ IBM Page Detailer that is available as a separate product.

## 17.1 Introduction

Performance of an application must be a focus area throughout the project cycle. Traditionally performance testing has occurred late in the cycle, towards the end of the testing phase, or as part of the deployment process. This does not leave much time to identify and fix any performance issues, particularly if significant architecture changes are required. It is much more difficult and time consuming to improve performance in the later phases of the project cycle than to ensure the application performs adequately from the beginning. To support this, IBM WebSphere Studio Application Developer includes facilities for profiling and measuring performance of Web applications. These tools can be used during the coding phase of the development cycle to identify problems earlier. Two of the tools that are available to measure and analyze performance are the *profiling tools (Profiler)* that have been incorporated into IBM WebSphere Studio Application Developer, and *Page Detailer*, which is downloadable from IBM alphaWorks® at:

<http://www.alphaworks.ibm.com/tech/pagedetailer>

The profiling tools help with the analysis of the runtime behavior of aspects of the application, such as execution time and flows and memory usage. The Page Detailer assists with the analysis of the behavior of browsers when accessing the application.

See 17.2, “The Profiler (profiling tools)” on page 760 and 17.3, “IBM Page Detailer” on page 792 for details on these products.

The application we have been using to demonstrate these tools is the Trade3 sample application. See 7.7, “Installing and configuring Trade3.1” on page 298 for more information about Trade3.

## 17.2 The Profiler (profiling tools)

The profiling tools incorporated into IBM WebSphere Studio Application Developer V5.1.1 can be used to collect and display data relating to the runtime behavior of the application and provide insight into how the Java Virtual Machine (JVM) is executing the application. The tools work in conjunction with the Agent Controller, which is a program separate from the base IBM WebSphere Studio Application Developer installation. We use the term Profiler many times throughout this chapter for the profiling tools.



## 17.2.1 Overview

The profiling tools can be used to analyze object creation and garbage collection behavior, execution sequences, thread interaction, and object references. This will provide an indication of which parts of the application tuning activities you should focus on.

The tools can display profiling data in a number of graphical and statistical (tabular) views that highlight different aspects of the runtime behavior of the application. The information that is provided by the profiling tools includes:

- ▶ Graphical profiling views:
  - Execution Flow view and table
  - Method Invocations view and table
- ▶ Sequence diagrams:
  - Object Interactions
  - Class Interaction
  - Thread Interactions
  - Process Interactions
  - Host Interactions
- ▶ Statistical profiling views:
  - Package statistics
  - Class statistics
  - Method statistics
  - Instance statistics

**Note:** Due to performance and usability reasons, the Heap and Object Reference graphical views that were available in IBM WebSphere Studio Application Developer V5.0 have been removed and are no longer available in IBM WebSphere Studio Application Developer V5.1.1. This current version provides improved statistic tables that contain heap and execution profiling data. These tables used in conjunction with the Object References Table display all the information that was previously provided by the dropped graphical views.

Details of these views are provided in 17.2.5, “Profiler views” on page 776. The information obtained from various views provides different perspectives about the application behavior and performance. By combining this information you can get a total picture of the performance of your application. The different views are linked together, so that if you highlight an item in one view and move to another, a corresponding item will be highlighted. This makes it easier to collect all the information about particular classes, objects, or methods.

We recommend that you go through the hands on guides for the Profiler available in IBM WebSphere Studio Application Developer V5.1.1. Select **Help -> Help Contents -> Testing -> Log and Trace Analyzer Features -> Profiling Tool**. This will help you develop and hone your profiling skills.

## 17.2.2 IBM Agent Controller

As mentioned before, the profiling tools work in conjunction with the IBM Agent Controller, which is a separate program from the WebSphere Application Server installation. Thus you need to make sure that the Agent Controller has been installed on the host where you want to monitor the process.

The Agent Controller is a daemon process that enables client applications to launch host processes and interact with agents that coexist within host processes. The Agent Controller interacts with the following components:

- ▶ Host process

The host process contains the application that is to be profiled. This can be a stand-alone Java process or a WebSphere Application Server instance (including an instance running in an IBM WebSphere Studio Application Developer test environment). An Agent Controller must be executing on the same machine as the host controller.

- ▶ Agent

The Java Profiling Agent is a library that provides services to the host process to capture and record the behavior of a Java application, and makes this data available to attached clients. The agent uses the Java Virtual Machine Profiler Interface (JVMPi) to interface with the JVM. A host process can have one or more agents currently running within it.

- Java Profiling Agent

The Java Profiling Agent can collect detailed information about any application executing in a JVM. The examples in this chapter use the Java Profiling Agent.

- J2EE Request Profiler

The J2EE Request Profiler is an agent that operates in a WebSphere Application Server process and that collects data relating to the execution of the Web application, such as execution of EJBs and servlets.

Distributed environments can be profiled by connecting to J2EE Request Profiler agents running on different nodes. The J2EE Request Profiler only provides a subset of the information available from the Java Profiling Agent, since it only looks at the execution of the enterprise components, and not at Java objects and methods that are called from the EJBs and servlets. The J2EE Request Profiler does, however, have the ability to

combine data from more than one WebSphere instance, allowing you to analyze and measure the behavior and performance of a distributed application.

► Client

A client is a local or remote application that retrieves and displays data that is collected by an agent. A single client can be attached to many agents at once, and can connect to Agent Controllers on local or remote machines. All communication between the client and agents occurs via the Agent Controller. A profiler client is provided with IBM WebSphere Studio Application Developer. The profiling functionality in IBM WebSphere Studio Application Developer V5.1.1 can be accessed from the *Profiling and Logging Perspective*.

In this chapter, we discuss the profiling of an application running on WebSphere Application Server. Similar techniques can be used to remotely profile applications running in a WebSphere Studio test environment.

Before attempting to profile an application, ensure that the Agent Controller has been started. This can be checked from the Services program in Windows 2000. If the status of the IBM Agent Controller service is not “Started”, then it can be started in Windows 2000 by clicking the **Start Service** button as shown in Figure 17-1.

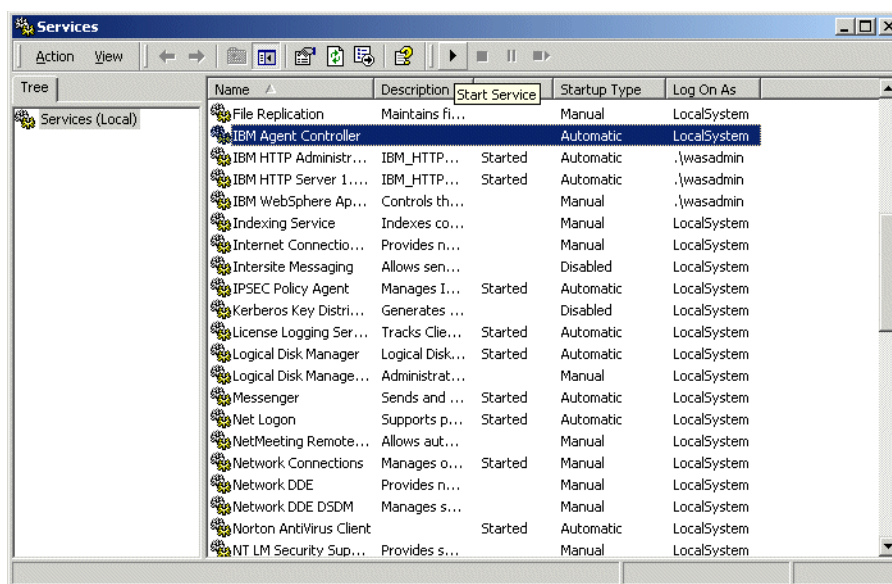


Figure 17-1 Starting the Agent Controller

### 17.2.3 Profiler configuration

We will profile the Trade3 application by attaching to a remote process. This allows us to take more accurate measures of time, without being affected by context switching. It also saves resources on the development machine. See “Profiler configuration for WebSphere Studio test environment” on page 765 for information on how to profile an application in the WebSphere Studio test environment.

#### Profiler configuration for a remote process

The remote server should have been started with profiling enabled. To configure your application server for profiling, do the following:

1. In the Administrative Console select **Servers -> Application Servers -> <AppServer> -> Process Definition**. Enter **-XrunpiAgent** into the Executable arguments field and click **OK**.
2. Go to **Servers -> Application Servers -> <AppServer> -> Process Definition**. Select **Java Virtual Machine** from the Additional Properties pane. Enter **-XrunpiAgent -DPD\_DT\_ENABLED=true** into the Generic JVM arguments field and click **OK**.
3. Save the configuration and restart the server(s).
4. The next step is to attach to a Remote Java process for the server instance:
  - a. In WebSphere Studio Application Developer, switch to the **Profiling and Logging** perspective. How to switch to the Profiling and Logging Perspective is shown in Figure 17-2 on page 765.
  - b. Select **Profile -> Attach -> Remote Process**. This option can also be obtained by right-clicking in the profiling monitor.
5. Go to “Generic Profiler configuration (local and remote environments)” on page 767.

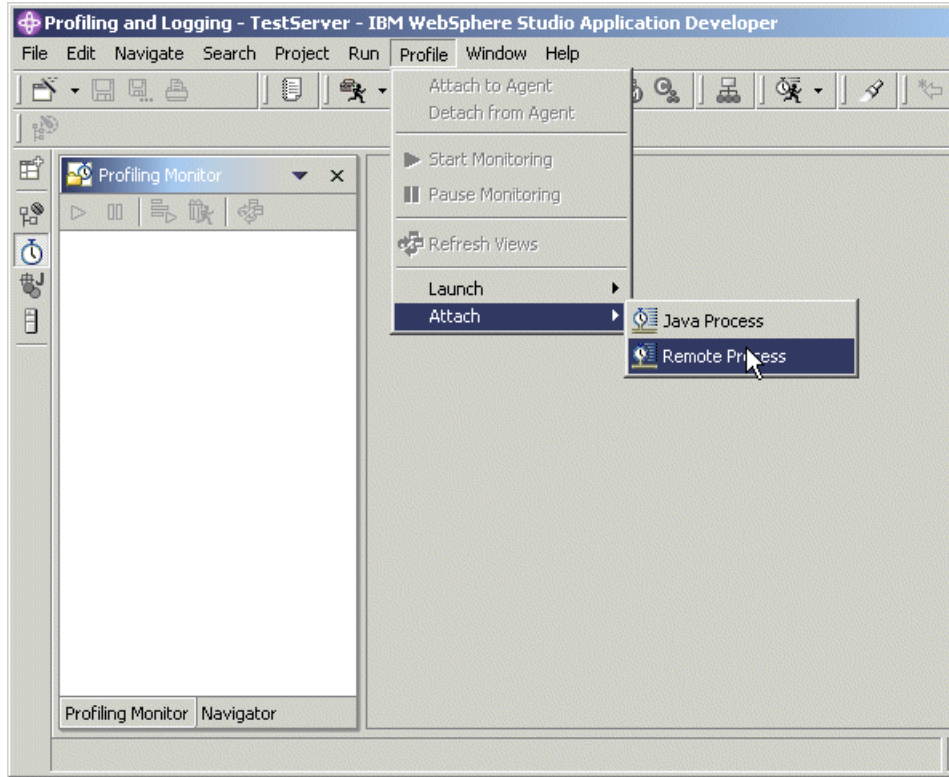


Figure 17-2 Profiling and Logging Perspective

### Profiler configuration for WebSphere Studio test environment

To profile an application in IBM WebSphere Studio Application Developer V5.1.1, the server in the test environment should have been started in profiling mode. From the Server or J2EE Perspective, right-clicking the server displays a menu that includes the profiling option. This is shown in Figure 17-3 on page 766. As a prerequisite, you should have previously deployed the application to the server and configured any server resources required. We suggest that you perform some basic testing of the application in IBM WebSphere Studio Application Developer prior to using the profiling tools, to ensure that the application is operating correctly.

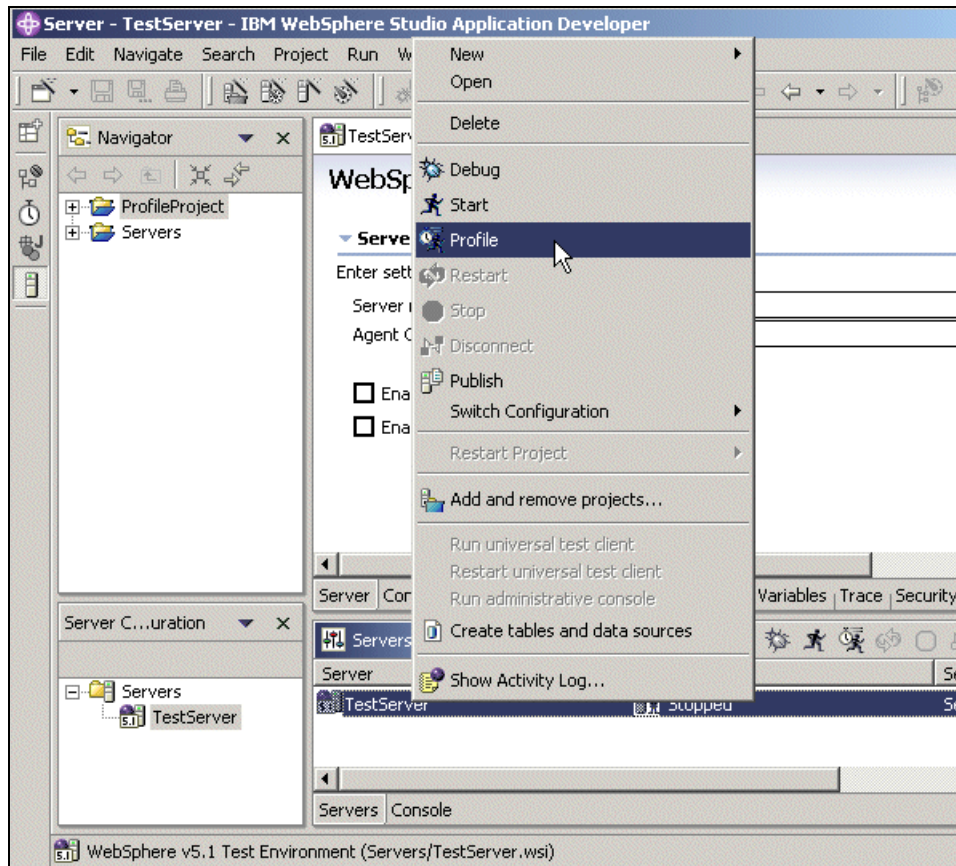


Figure 17-3 Starting a server in profiling mode

Once the server has been started, go to the Profiling and Logging Perspective. It can be accessed by clicking **Open a Perspective -> Profiling and Logging** as shown in Figure 17-4 on page 767.

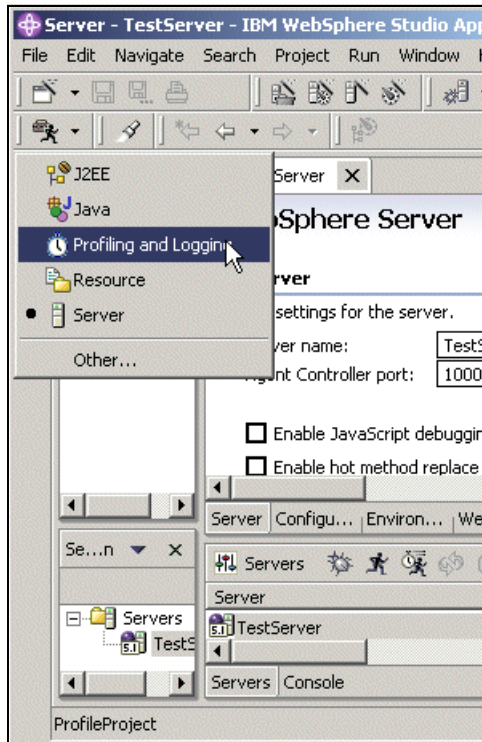


Figure 17-4 Selecting the Profiling and Logging Perspective

To profile a local application instead of a remote one, select **Profile -> Attach -> Java Process**.

## Generic Profiler configuration (local and remote environments)

The remainder of this procedure is the same for both local and remote profiling. A window is displayed to select the Java process to attach to. The process ID is the same as for the WebSphere Application Server instance and it can be found in the .pid file created in the server root, when the server is started.

There are two agents available: Java Profiling Agent and J2EE Request Profiler.

1. To obtain detailed information about the runtime behavior of the application, select the **Java Profiling Agent** option as shown in Figure 17-5 on page 768. The J2EE Request Profiler will provide higher-level overview data about the application. It is also possible to select both agents from this window.



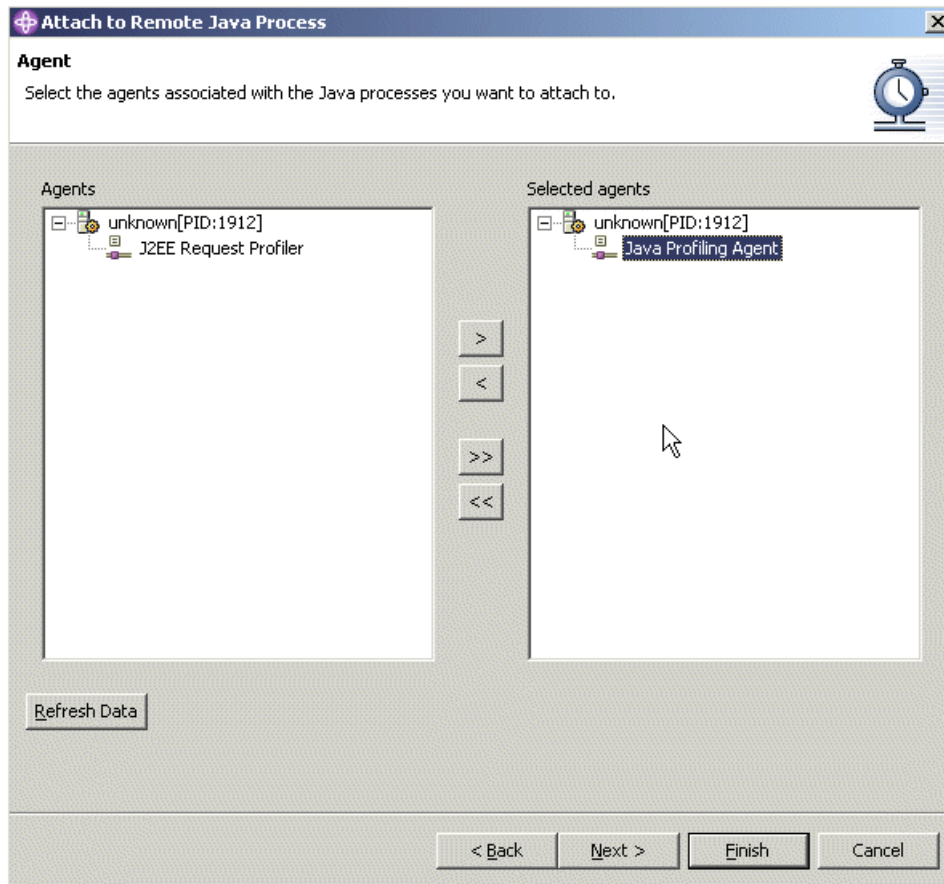


Figure 17-5 Select the agents associated with the Java processes

**Important:** When studying the execution flow of an application, we recommend that the J2EE Request Profiler be used. On the other hand, to obtain a detailed information relating to instance sizes, we suggest using the Java Profiling Agent with **Select instance level information** checked as shown in Figure 17-8 on page 771.

2. Click the **Next** button to obtain the window for specifying the profiling project and the monitor for saving the profiler data. Usually the default values can be kept.

A feature that has been introduced since WebSphere Studio Application Developer V5.0 is the ability to redirect output to a file. This is especially useful in situations where the information generated from the application



being monitored will exceed the amount of memory that WebSphere Studio Application Developer has been allocated or that is available on the system.

3. Click the **Next** button once again to get to the Profiling Filters window, as displayed in Figure 17-6. This allows you to specify the classes for which profiling information should be gathered.

There are a number of predefined filter sets available. Each of the filter sets specifies a set of rules to apply to determine if methods from particular classes and packages should be excluded or included. When profiling with the WebSphere test environment, we suggest you select the WebSphere J2EE filter as a good starting point, and customize it as required.

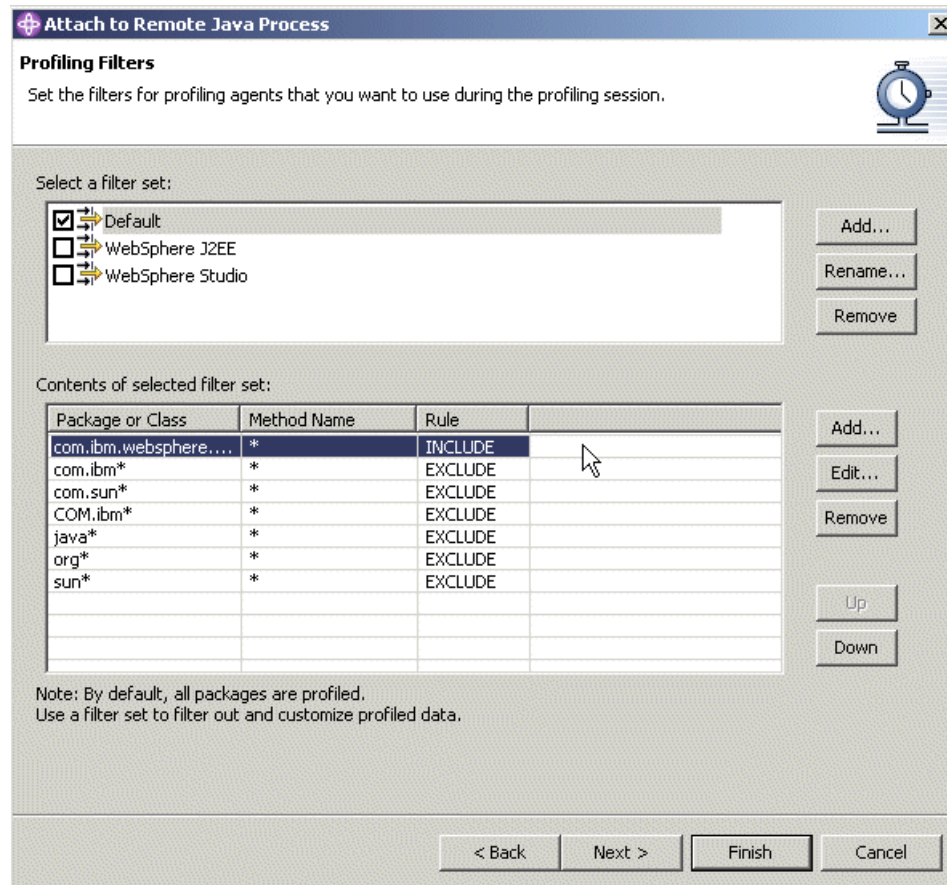


Figure 17-6 Profiling Filters

4. New rules can be added by clicking the **Add** button. This displays the window shown in Figure 17-7 on page 770. Each filter has three attributes - a package or class name, a method name and a flag to specify if the filter is for inclusion

or exclusion of particular data. Wildcards (\*) can be used when specifying the package or class and method names. As well as adding new filters, existing filters can be edited or removed, and the ordering of filters changed from the Profiling Filters window. The ordering is important as it specifies the precedence of the filter rules. If a rule for a subpackage differs from the rest of the package, the subpackage rule must appear first in the list. Completely new filter sets can also be created, and existing filter sets can be removed or renamed.

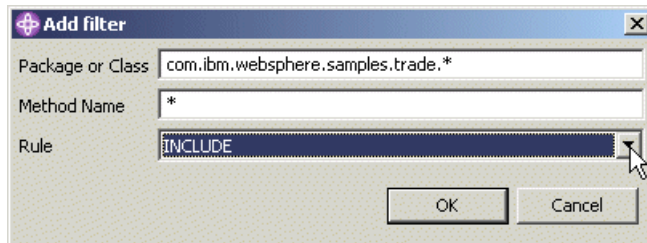


Figure 17-7 Specifying a new filter

When specifying filter sets remember that collecting data about too many classes will slow down the Profiler and can make it more difficult to analyze the results due to the volume of data. Normally the standard Java and WebSphere runtime classes are excluded, as the aim is to profile the application code.

5. Click **Next** again to display the Profiling Options window as shown in Figure 17-8 on page 771. It allows specification for whether execution flow information should be captured.

Make sure that the **Collect instance-level information** check box is selected.

The "Collect boundary classes excluded by the filter" check box allows you to specify whether profiling data should be collected about classes that are not included in the filter list, but that are directly referenced by classes that are in the filter list.

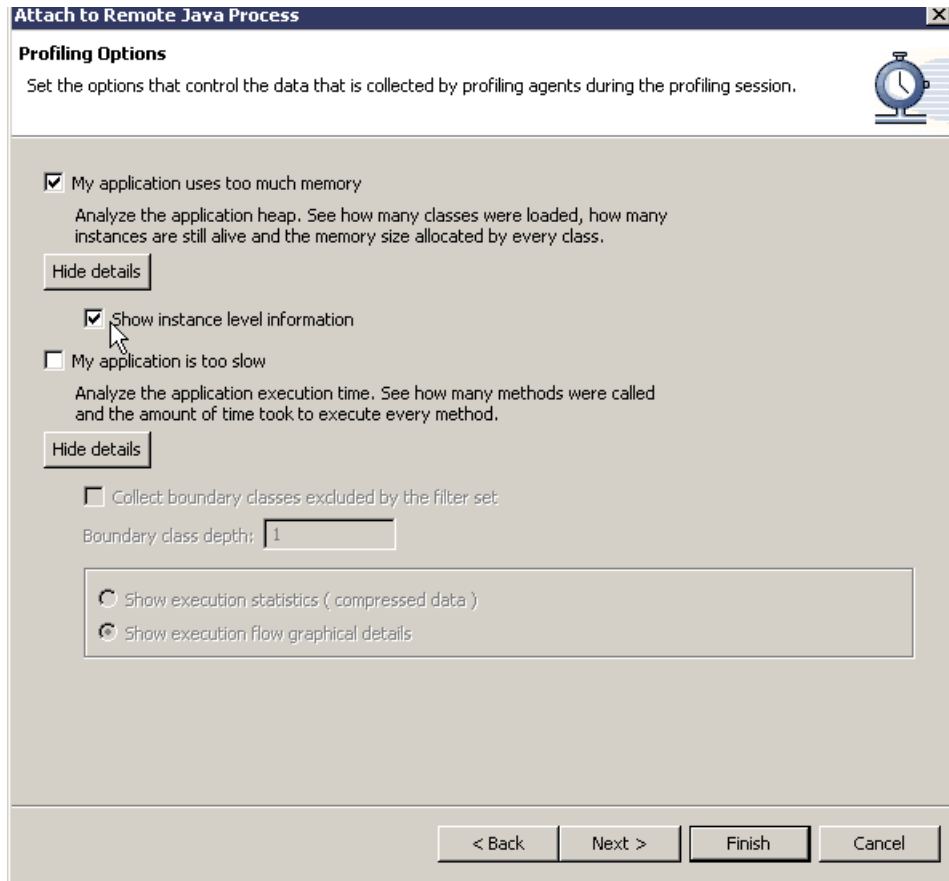
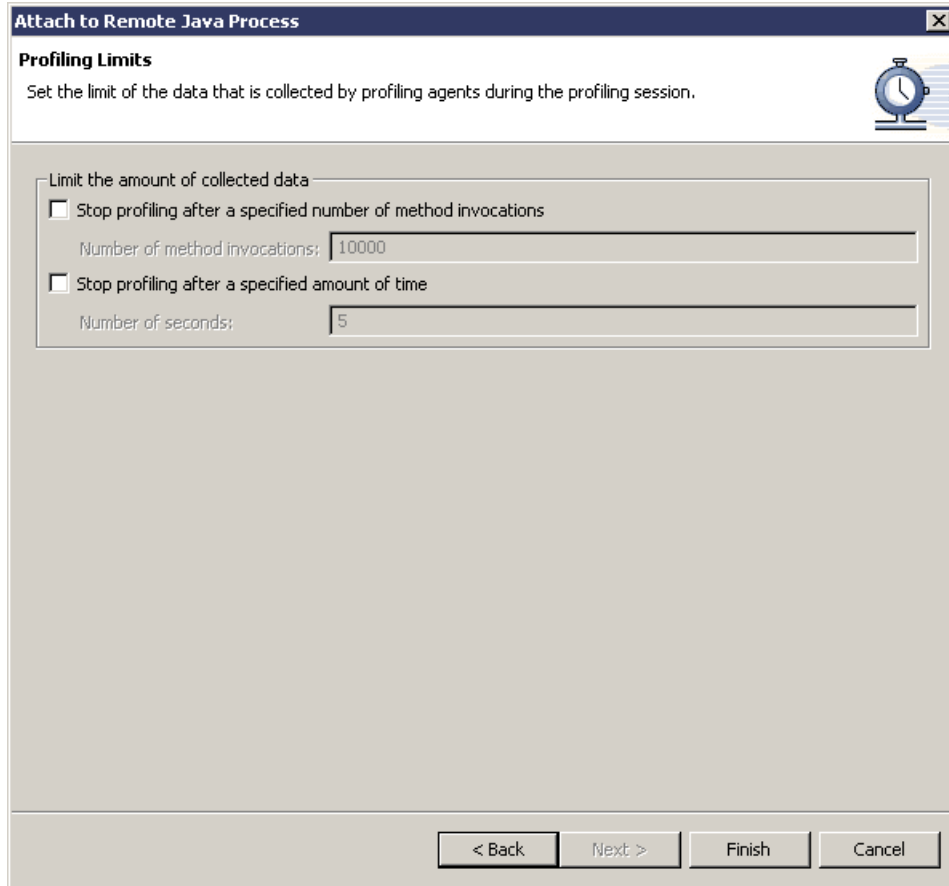


Figure 17-8 Profiling Options

6. Click **Next** to display the window shown in Figure 17-9 on page 772 where you can limit the amount of data collected by the Profiler. It allows you to instruct the Profiler to stop profiling after a particular number of method invocations, or after a specified time has elapsed.



The image shows a Windows-style dialog box titled "Attach to Remote Java Process". Inside, there is a section titled "Profiling Limits" with a subtitle "Set the limit of the data that is collected by profiling agents during the profiling session." and a clock icon. Below this, a group box labeled "Limit the amount of collected data" contains two unchecked checkboxes. The first checkbox is "Stop profiling after a specified number of method invocations" with a text input field set to "10000". The second checkbox is "Stop profiling after a specified amount of time" with a text input field set to "5". At the bottom of the dialog are four buttons: "< Back", "Next >", "Finish", and "Cancel".

**Attach to Remote Java Process**

**Profiling Limits**  
Set the limit of the data that is collected by profiling agents during the profiling session.

Limit the amount of collected data

☐ Stop profiling after a specified number of method invocations  
Number of method invocations: 10000

☐ Stop profiling after a specified amount of time  
Number of seconds: 5

< Back   Next >   Finish   Cancel

Figure 17-9 Limit the amount of data collected

7. Click **Finish** to complete the procedure for attaching to the (remote) Java process.

### Start data collection

At this point, the Profiler is configured but not collecting data. To start collecting data, select the agent in the Profiling Monitor pane, and choose **Profile -> Start Monitoring** from the menu as shown in Figure 17-10 on page 773.

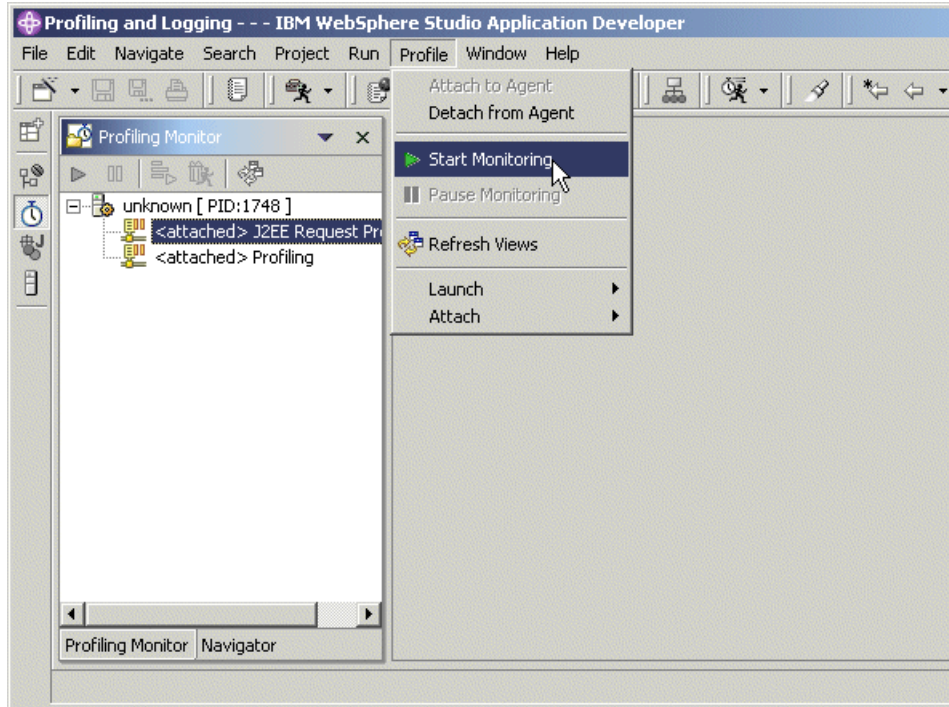


Figure 17-10 Start monitoring

Once monitoring has started, you are ready to start profiling the application.

## Configuration settings for profiling

Note that configuration settings for profiling can be accessed from the **Window** -> **Preferences** menu of the Workbench. They can be found on the Profiling and Logging page of the Preferences window as displayed in Figure 17-11:

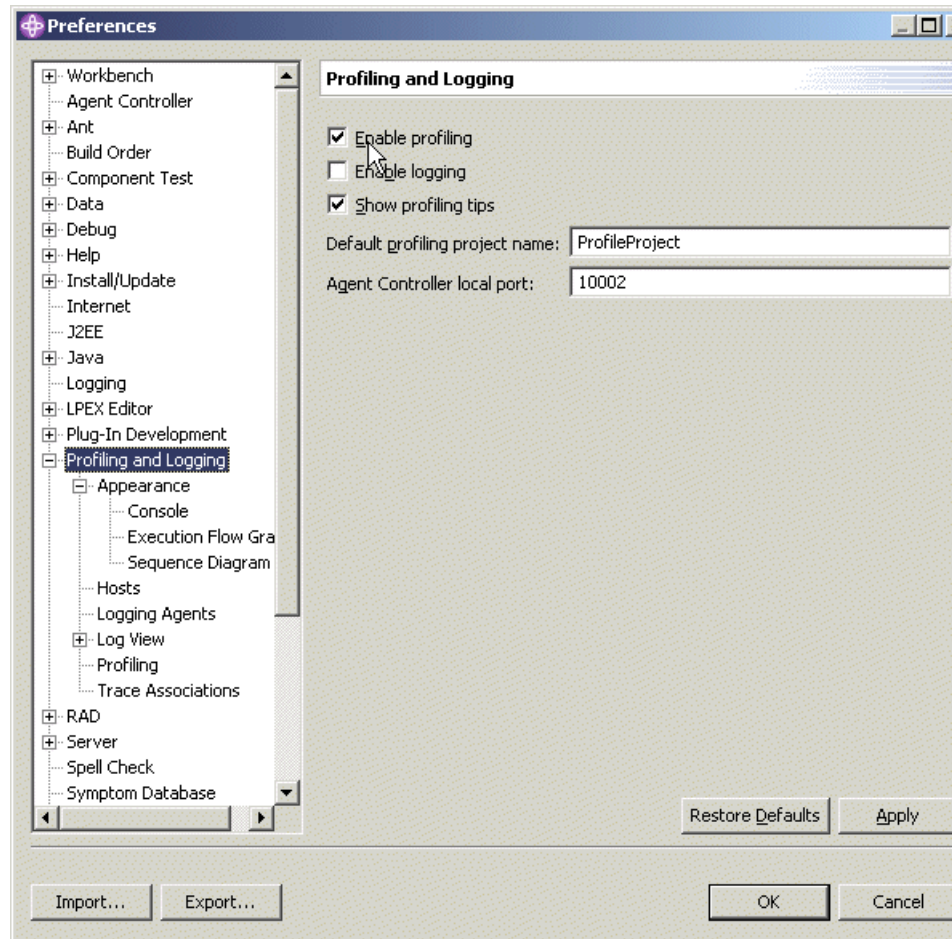


Figure 17-11 Preferences for profiling

- ▶ Ensure that the **Enable profiling** check box is selected.
- ▶ If the **Show profiling tips** check box is selected, tips will be displayed periodically during your profiling session.
- ▶ The Default profiling project name can be changed if desired. This specifies under which project in IBM WebSphere Studio Application Developer the profiling data will be stored. Although the Agent Controller local port can be

changed, this is usually not necessary. The port specified here must be the same as in the configuration for the Agent Controller. The default is 10002.

- ▶ Select **Profiling and Logging -> Profiling** for options to set default values for the profiling filters and options. Here you can limit the amount of data to be collected.
- ▶ Select **Profiling and Logging -> Appearance** to specify the format for Console, Execution Flow Graph and Sequence Diagram. Here you can also specify the resources that will appear in the Profiling Monitor window. As a minimum, the **Process**, **Profiling Agents** and **Logging agents** check boxes should be selected.
- ▶ Select **Profiling and Logging -> Appearance -> Sequence Diagram** to specify whether recursive calls should be displayed in sequence diagrams that are generated by the profiling tools. Although displaying recursive calls can complicate the sequence diagrams, displaying the recursive calls can be useful in understanding program execution.
- ▶ Select **Profiling and Logging -> Hosts** to specify a default list of hosts to be used when profiling a remote application.

## 17.2.4 Profiling the application

This section describes the tasks involved in profiling an application.

### Testing methodology

There are a few different approaches that can be taken when profiling a Web application. One approach is to profile individual pieces of functionality (possibly as part of the unit test process as they are being developed). Alternatively, a mixture of operations that represent typical user behavior could be executed, either manually or using a load testing tool such as Mercury LoadRunner, Rational® Robot, or Apache JMeter. Initially the testing can cover a broad range of scenarios, and then as particular areas of functionality are identified, more detailed profiling can be done for them. The Trade3 application includes a servlet “scenario” that simulates a set of users by executing a trade operation for a random user on each invocation of the servlet. A similar servlet or client program that exercises a broad range of application functionality may also be useful for other applications.

### Using the Profiler

Once the Profiler has been configured, the application can be accessed via the normal URL and the profiling data will be captured. In order for the captured data to be displayed in the Profiling Perspective, right-click in the Profiling Monitor view and select **Refresh Views**. This will update the data for all views, not just



the one that is currently displayed. Once the data has been displayed, the view will have to be refreshed again in order to display newly captured data.

## 17.2.5 Profiler views

As outlined in 17.2.1, “Overview” on page 761, there are a number of different views that can be displayed in the Profiling Perspective. To open a new view, right-click in the Profiling Monitor view and select one of the options from the Show View menu. Profiler views can also be displayed using the tool bar. The Profiler menu and toolbar are shown in Figure 17-12.

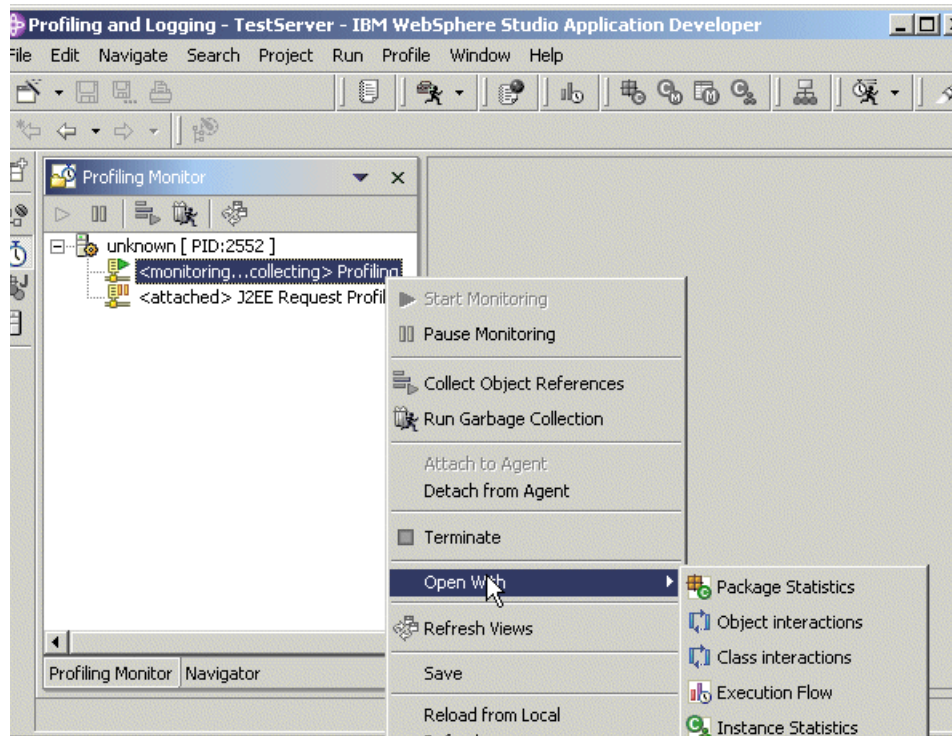


Figure 17-12 Profiler View menus

The choice of views available and their content depends on the context menu of the resource. For example, the Host Interaction View will be available from the context of the monitor. You can make the resources visible either from **Window -> Preferences** or from the perspective as shown in the Figure 17-13 on page 777.



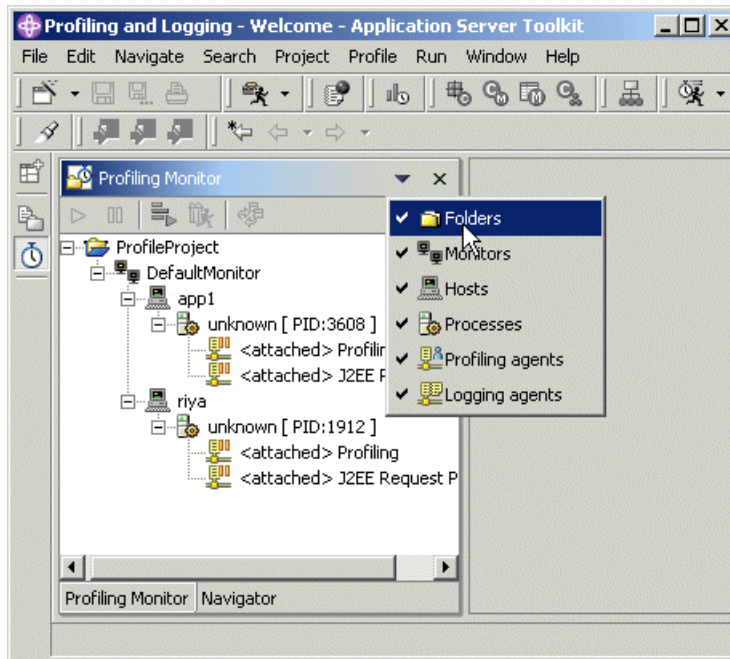



Figure 17-13 Selecting Resources



## Column information

Since many of the columns are available from more than one view, an explanation of the different columns available in the statistics views is provided here. The columns displayed in each view can be customized by right-clicking in the view and selecting **Choose Columns** from the menu.

- ▶ **Package**  
The name of the package.
- ▶ **Class Names**  
The name of the class.
- ▶ **Instance Name**  
Each object instance is given a unique name so that it can be easily identified. The naming convention is class name.id, where ID is a sequential ID for each class.
- ▶ **Method Names**  
The name of the method.

► Delta

When a refresh is performed as described in “Using the Profiler” on page 775, this symbol  is shown for any rows that are new since the last refresh of the view.

Individual cells whose values have changed since the last refresh are displayed with icons that indicate an increase  or decrease .

► Calls

When a row is displayed at the method level (in the class method or method statistics view), this column shows the number of times the method has been called. For rows that are displayed at a class level, the total number of calls for all methods of the class or class instance are displayed. When a row is displayed at a package level, calls to methods of classes belonging to the package are summed.

► Inherited Calls

Number of Calls plus the number of calls made to inherited methods.

► Base Time

The total time spent executing the method(s) for the current row. It does not include time spent in other methods that are called by this method.

► Inherited Base Time

Base Time plus the time spent in inherited methods.

► Average Base Time

The average time spent in each execution of the method(s).

► Cumulative Time

The time spent executing the method(s), including any methods that are called by it.

► Inherited Cumulative Time

Cumulative Time plus the time spent in inherited methods.

► Collected

The number of object instances that have been garbage collected.

► Live Instances

The number of object instances that currently exist.

► Total Instances

The number of object instances that have been created, including live instances and ones that have been garbage collected.

- Total Size

The size in bytes of all instances that were created for the selected package, class or method. It includes instances that have already been garbage collected.

- Active Size

The total size in bytes for all live object instances.

**Important:** All times displayed in the Profiler are execution clock times, and not CPU times. Hence execution times can be affected by context switching if other programs and processes are executing. We recommend that you shut down all unnecessary programs, services, etc., to minimize the impact of this. We also recommend that tests are performed more than once, since results may be affected by transient factors on the test machine and hence results from individual runs may not be representative of normal performance.

## Package Statistics Table

The Package Statistics Table displays the profiling data for the package with an ability to drill down to the class level (see Figure 17-14 on page 780).

This information allows you to get a high-level view of which packages are most frequently used, based on the frequency on which methods of classes belonging to the package are called and the number of instances of these classes. It also provides information on which packages use the most system resources in terms of execution time of methods for classes defined in the package and the memory used by object instances of the classes. It is possible to drill down to the class level for a particular package by clicking the + symbol displayed on the left of the package name. To reduce the amount of data being displayed, the output can be filtered based on the package name. These filters can include \* as a wildcard character. The output can be sorted based on any of the displayed columns by clicking the column name.

The information that is displayed is:

- Package
- Delta icon
- Total Instances
- Live Instances
- Collected
- Active Size
- Total Size
- Base Time (not displayed by default)
- Inherited Base Time (not displayed by default)
- Cumulative Time (not displayed by default)

- ▶ Inherited Cumulative Time (not displayed by default)
- ▶ Calls (not displayed by default)
- ▶ Inherited Calls (not displayed by default)

T

The screenshot shows the 'Profiling and Logging' window in IBM WebSphere Studio Application Developer. The title bar reads 'Profiling and Logging - - IBM WebSphere Studio Application Developer'. The menu bar includes 'File', 'Edit', 'Navigate', 'Search', 'Project', 'Run', 'Profile', 'Window', and 'Help'. The toolbar contains various icons for file operations, navigation, and profiling. The main window is titled 'Package Statistics - unknown [ PID:2624 ]'. It features a 'Filter:' text box and a 'Case-sensitive' checkbox. Below this is a table with the following data:

>Package	Total Instances	Live Instances	Collected	Total Size	Active Size
com.ibm.websphere.samples.trade.ejb	333	283	50	12084	11084
_EJ3RemoteCMPQuoteEJBHome	0	0	0	0	0
_EJ3RemoteCMPQuoteEJBHome...	0	0	0	0	0
_EJ3RemoteStatelessTradeEJB...	0	0	0	0	0
_EJ3RemoteStatelessTradeEJB...	0	0	0	0	0
_Trade_Stub	1	1	0	12	12
_TradeHome_Stub	1	1	0	12	12
AccountBean	0	0	0	0	0
AccountProfileBean	0	0	0	0	0
ConcreteAccountEJB_b5483e03	2	2	0	72	72
ConcreteAccountProfileEJB_3bd...	2	2	0	40	40
ConcreteHoldingEJB_e10e4ecb	6	6	0	216	216
ConcreteKeyGenEJB_11db8b94	0	0	0	0	0
ConcreteOrderEJB_725b53ca	1	1	0	36	36
ConcreteQuoteEJB_73b52d85	100	50	50	2000	1000
EJ3CMPAccountEJBHomeBean_...	0	0	0	0	0

Figure 17-14 Package Statistics Table

## Class Statistics Table

The Class Statistics Table (see Figure 17-15 on page 781) displays data for the classes with an ability to drill down to the method level.

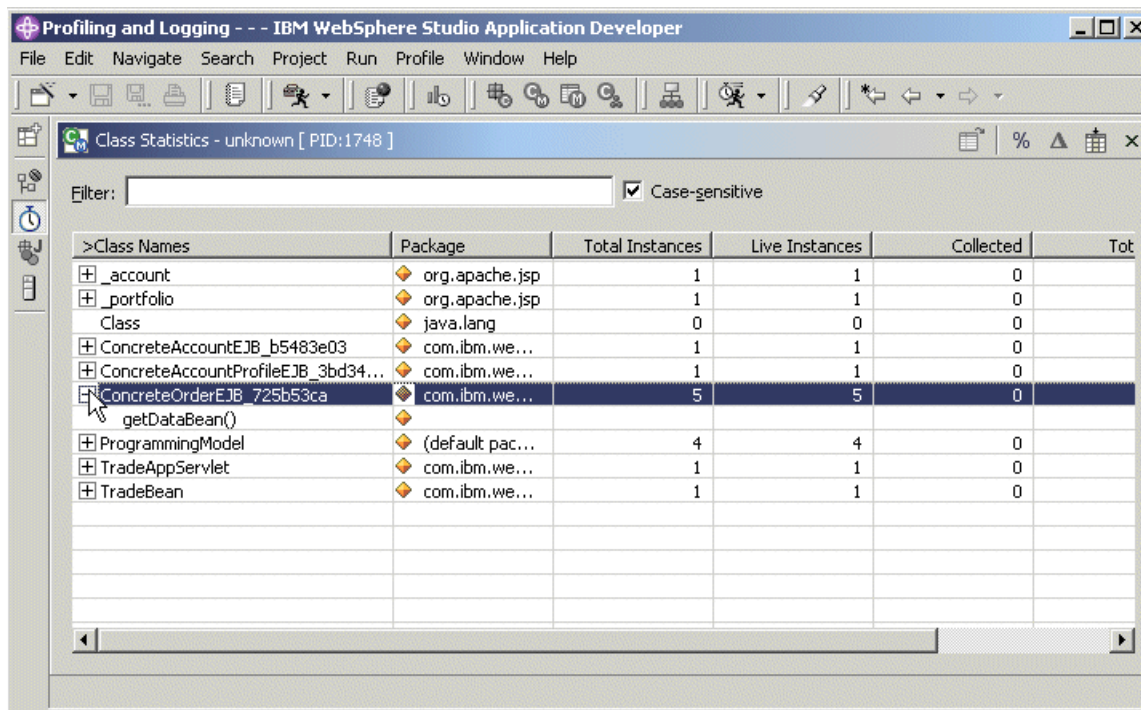
The data in this view provides the next lower level of details, after the Package Statistics. It shows which classes are most commonly used, the execution time for methods and the size of the objects. This makes it easier to identify particular classes that require further investigation.

The information that is displayed is:

- ▶ Class Names
- ▶ Package
- ▶ Delta icon
- ▶ Total Instances
- ▶ Live Instances
- ▶ Collected

- ▶ Active Size
- ▶ Total Size
- ▶ Base Time (not displayed by default)
- ▶ Inherited Base Time (not displayed by default)
- ▶ Cumulative Time (not displayed by default)
- ▶ Inherited Cumulative Time (not displayed by default)
- ▶ Calls (not displayed by default)
- ▶ Inherited Calls (not displayed by default)

**Note:** Information related to instance sizes is available only if you open the view with the Java Profiling Agent.



The screenshot shows the 'Profiling and Logging' window in IBM WebSphere Studio Application Developer. The title bar indicates the application is running on PID:1748. The 'Class Statistics' tab is active, displaying a table with columns: >Class Names, Package, Total Instances, Live Instances, Collected, and Tot. The table lists several classes, with 'ConcreteOrderEJB\_725b53ca' selected, showing 5 total and live instances. A filter box and a 'Case-sensitive' checkbox are also visible.

>Class Names	Package	Total Instances	Live Instances	Collected	Tot
+ _account	org.apache.jsp	1	1	0	
+ _portfolio	org.apache.jsp	1	1	0	
Class	java.lang	0	0	0	
+ ConcreteAccountEJB_b5483e03	com.ibm.we...	1	1	0	
+ ConcreteAccountProfileEJB_3bd34...	com.ibm.we...	1	1	0	
+ ConcreteOrderEJB_725b53ca	com.ibm.we...	5	5	0	
getDataBean()					
+ ProgrammingModel	(default pac...	4	4	0	
+ TradeAppServlet	com.ibm.we...	1	1	0	
+ TradeBean	com.ibm.we...	1	1	0	

Figure 17-15 Class Statistics Table

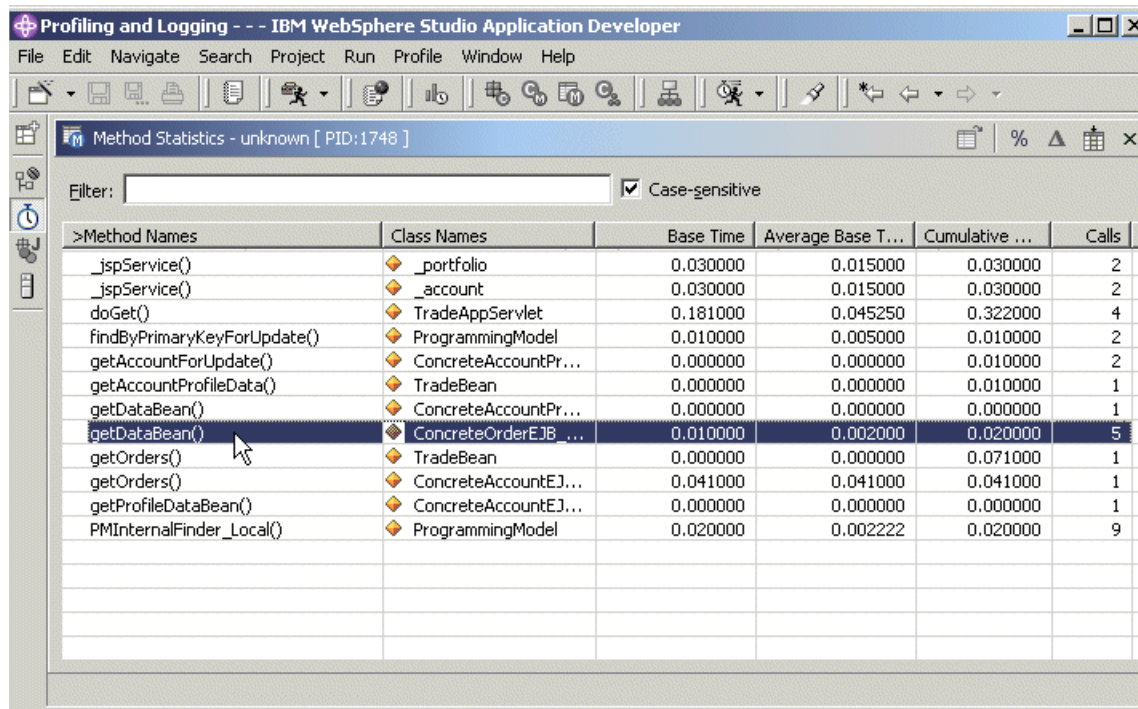
## Method Statistics Table

The Method Statistics Table (see Figure 17-16 on page 782) allows you to view profiling data about particular methods. This table is useful in identifying the time consuming methods in an application.

It allows you to determine which particular methods are most frequently executed and those that have the longest execution time. Once these methods have been identified, effort can be focused on optimizing these methods. After making changes, the application can be profiled again and differences in performance measured. This optimization process is likely to be iterative, but the class method statistics allows you to approach this in a systematic and scientific way. However, in order for the test results to be repeatable and valid comparisons made, the test environment should be the same for each test. We suggest that all other applications be closed, and that each individual test be run multiple times to reduce the probability of getting spurious results.

It contains the following information:

- ▶ Class Names
- ▶ Method Names
- ▶ Delta icon
- ▶ Package (not displayed by default)
- ▶ Base Time
- ▶ Average Base Time
- ▶ Cumulative Time
- ▶ Calls



The screenshot shows the 'Profiling and Logging' window in IBM WebSphere Studio Application Developer. The title bar reads 'Profiling and Logging - - - IBM WebSphere Studio Application Developer'. The menu bar includes File, Edit, Navigate, Search, Project, Run, Profile, Window, and Help. The toolbar contains various icons for file operations, navigation, and profiling. The main window is titled 'Method Statistics - unknown [ PID:1748 ]'. Below the title bar is a filter input field and a 'Case-sensitive' checkbox. The table below lists method statistics.

>Method Names	Class Names	Base Time	Average Base T...	Cumulative ...	Calls
_jspService()	_portfolio	0.030000	0.015000	0.030000	2
_jspService()	_account	0.030000	0.015000	0.030000	2
doGet()	TradeAppServlet	0.181000	0.045250	0.322000	4
findByPrimaryKeyForUpdate()	ProgrammingModel	0.010000	0.005000	0.010000	2
getAccountForUpdate()	ConcreteAccountPr...	0.000000	0.000000	0.010000	2
getAccountProfileData()	TradeBean	0.000000	0.000000	0.010000	1
getDataBean()	ConcreteAccountPr...	0.000000	0.000000	0.000000	1
getDataBean()	ConcreteOrderEJB ...	0.010000	0.002000	0.020000	5
getOrders()	TradeBean	0.000000	0.000000	0.071000	1
getOrders()	ConcreteAccountEJ...	0.041000	0.041000	0.041000	1
getProfileDataBean()	ConcreteAccountEJ...	0.000000	0.000000	0.000000	1
PMInternalFinder_Local()	ProgrammingModel	0.020000	0.002222	0.020000	9

Figure 17-16 Method Statistics Table



## Instance Statistics Table

The Instance Statistics Table (see Figure 17-17) allows you to view data about particular instances of classes. The Instance Statistics Table contains the following information:

- ▶ Instance Name
- ▶ Class Names
- ▶ Package
- ▶ Delta icon
- ▶ Total Instances
- ▶ Live Instances
- ▶ Collected (not displayed by default)
- ▶ Active Size
- ▶ Total Size
- ▶ Base Time (not displayed by default)
- ▶ Inherited Base Time (not displayed by default)
- ▶ Cumulative Time (not displayed by default)
- ▶ Inherited Cumulative Time (not displayed by default)
- ▶ Calls (not displayed by default)
- ▶ Inherited Calls (not displayed by default)

The screenshot shows the IBM WebSphere Studio Application Developer interface. The 'Profiling Monitor' window is open, displaying a tree view of the profiling data. The 'Instance Statistics - unknown [ PID:2724 ]' window is also open, showing a table of instance statistics. The table has columns for Class Names, Package, Total Instances, and Live Instances. The row for 'ConcreteQuoteEJB' is highlighted, showing 100 total instances and 73b52d85 live instances. The 'Filter' field is empty, and the 'Case-sensitive' checkbox is checked.

>Class Names	Package	Total Instances	Live I
+ _account	org.apache.jsp	1	
+ _displayQuote	org.apache.jsp	1	
+ _marketSummary	org.apache.jsp	1	
+ _portfolio	org.apache.jsp	1	
+ _quote	org.apache.jsp	1	
+ _tradehome	org.apache.jsp	1	
+ Class	java.lang	0	
+ ConcreteQuoteEJB 73b52d85	com.ibm.we...	100	73b52d85
+ ProgrammingModel	(default pac...	1	
+ TradeAppServlet	com.ibm.we...	1	
+ TradeBean	com.ibm.we...	1	

Figure 17-17 Instance Statistics Table

In this view, the Collected column displays the number of instances of the selected package, class, or method, that were removed during garbage collection.

From this table, it is possible to drill down to individual instances of particular classes by clicking the + symbol displayed on the left of the Class Name. By drilling down to individual instances, problems that may not be apparent just by looking at average or total values may be identified.

For example, particular instances of classes may be abnormally large. If this occurs, more details can be found by looking at the object references for the instance. To display this, right-click the instance and select **Show Object References** from the menu.

In general, determining why an object (including referenced objects) in Java requires a large amount of memory can be difficult, because the size of the object itself may be small, but there may be a set of references that eventually lead to a large object. This is a powerful tool to aid in identifying the root cause of excessive memory usage. As discussed in 18.2, “Memory” on page 805, minimizing memory usage is one of the key mechanisms for improving the performance of a Java application.

## Object References Table

The Object References Table (Figure 17-18 on page 786) displays the references from one object to another. This table replaces the graphical Object Reference View of the previous version.

The Object References Table can be extremely useful in optimizing the memory utilization of your application. It contains a tree structure that you can traverse to narrowing down to the source of memory leaks. By drilling down to individual instances, problems that may not be apparent just by looking at average or total values may be identified. By default it shows the instances that hold a reference to the selected instance. However, if you check **Show Reference To** then it shows all the references to other instances held by the selected instance.



**Tips:**

- ▶ You can choose to view the references **to** other objects or **by** other objects.
- ▶ You must take a dump of the heap by running **Collect Object References** from the context menu of the active process before you display the information in Object References Table.
- ▶ You can identify memory leaks in a particular unit of work by taking a dump just before triggering your transaction and then again at the end of the transaction. The objects that could not be collected between the two dumps will be labelled as new objects.

The Object References Table contains the following information:

- ▶ Show Reference By Class Name
- ▶ Show Reference To Class Name (not shown by default)
- ▶ Package
- ▶ Size
- ▶ Number of References
- ▶ Details

**Note:** Please note that the value shown in the Size column depends on the level of information you are looking at. On the first level, the sum for the corresponding classes is shown. When drilling down, the corresponding size of the level object is shown.

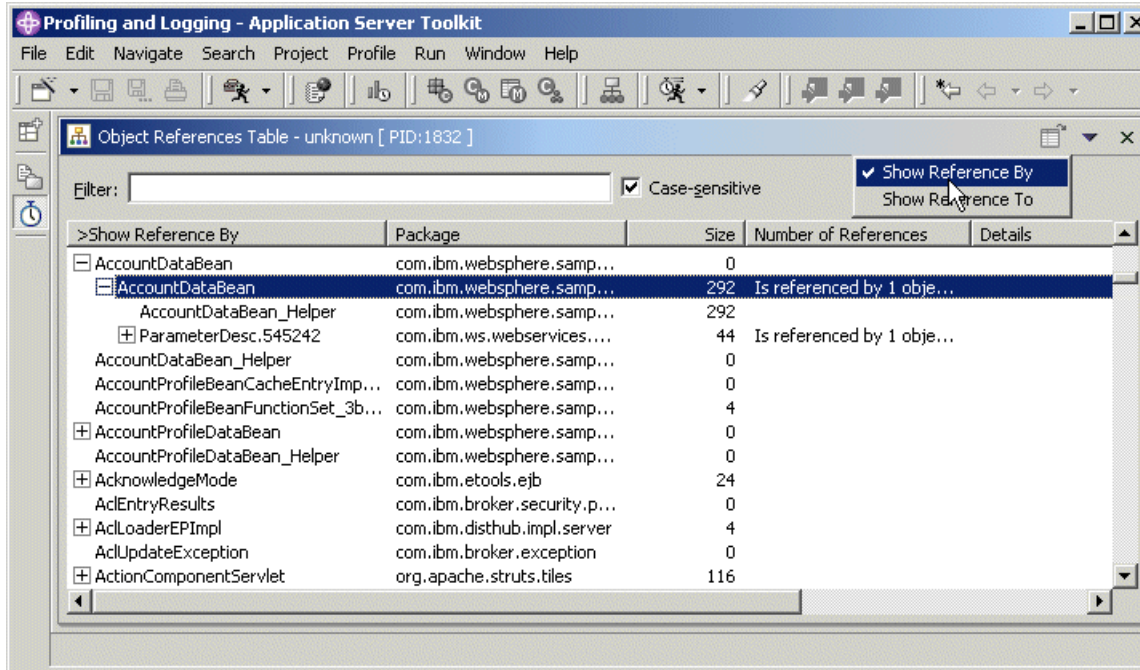


Figure 17-18 Object References Table

## Execution Flow View

The Execution Flow View visually displays the method calls performed by the application being profiled. In order to reduce the amount of information being displayed to a manageable level, we recommend that you use the J2EE Request Profiler Agent for capturing data to be viewed in the Execution Flow View. The Execution Flow View is another way of determining which methods are executed most frequently, and hence are good candidates for optimization. An example of the Execution Flow View is provided in Figure 17-19 on page 787.

The information that is displayed pictorially in the Execution Flow View can also be displayed in a hierarchical tabular form as shown in Figure 17-20 on page 788. In some cases it may be easier to navigate through the data in this view, because it is possible to expand and collapse particular parts of the execution sequence. From this view, you can right-click a particular method invocation to display the Method Invocation View for that method.

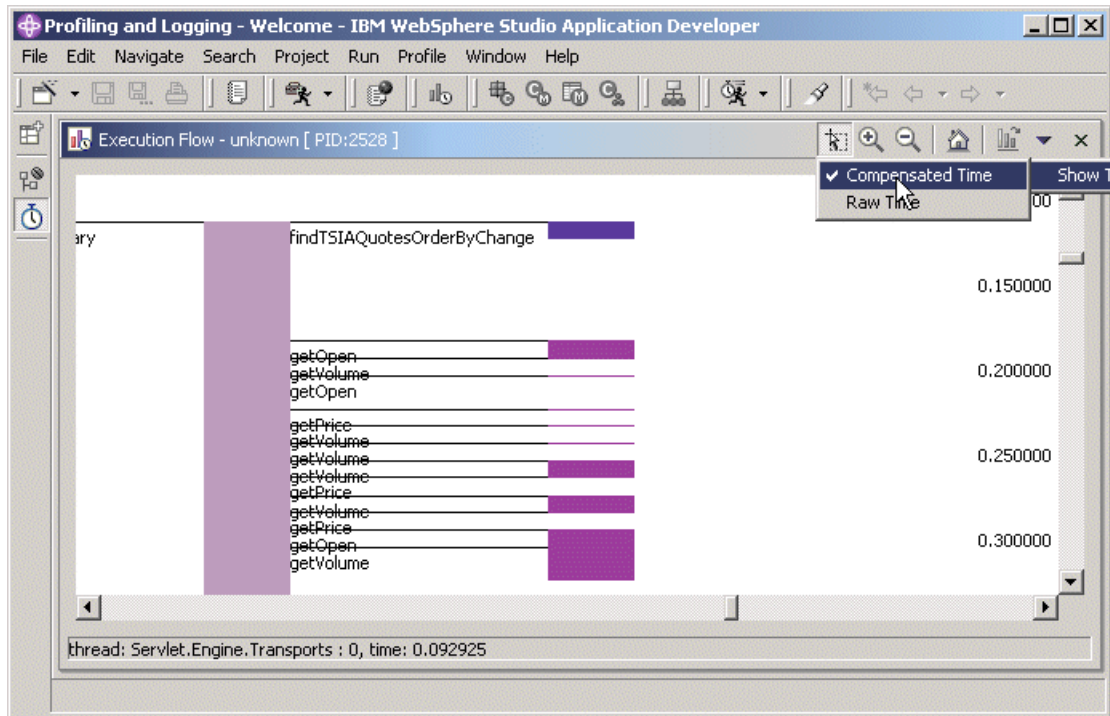


Figure 17-19 Execution Flow View

Profiling and Logging - Welcome - IBM WebSphere Studio Application Developer

File Edit Navigate Search Project Run Profile Window Help

Execution Flow Table - unknown [ PID:2528 ]

Filter:  ☒ Case-sensitive

Thread Names	Instance Name	>Start Time	Cumulative Time
Servlet.Engine.Transports : 0[2528]			
doGet	TradeAppServlet.4	-0.040000	1.172000
jspService	_tradehome.8	0.090000	1.042000
jspService	_marketSummary.12	0.100000	1.032000
getMarketSummary	TradeBean.16	0.110000	0.922000
findTSIAQuotesOrd...	ProgrammingModel.20	0.110000	0.010000
getPrice	ConcreteQuoteEJB_73b52d85.25	0.180000	0.000000
getOpen	ConcreteQuoteEJB_73b52d85.25	0.180000	0.010000
getVolume	ConcreteQuoteEJB_73b52d85.25	0.190000	0.000000
getPrice	ConcreteQuoteEJB_73b52d85.35	0.190000	0.000000
getOpen	ConcreteQuoteEJB_73b52d85.35	0.190000	0.000000
getVolume	ConcreteQuoteEJB_73b52d85.35	0.190000	0.000000
getPrice	ConcreteQuoteEJB_73b52d85.42	0.190000	0.000000
getOpen	ConcreteQuoteEJB_73b52d85.42	0.200000	0.000000
getVolume	ConcreteQuoteEJB_73b52d85.42	0.200000	0.000000
getPrice	ConcreteQuoteEJB_73b52d85.49	0.200000	0.000000
getOpen	ConcreteQuoteEJB_73b52d85.49	0.200000	0.000000

Figure 17-20 Execution Flow Table

## Object and Class Interaction (Sequence Diagram) views

The Object Interaction View and Class Interaction Views (also labeled as Sequence Diagram Views) allow analysis for program execution by displaying UML (Unified Modelling Language) sequence diagrams.

Normally these diagrams are produced statically during the analysis or design phase of a project, but IBM WebSphere Studio Application Developer V5.1.1 allows you to dynamically generate these diagrams to see how the program is really working, rather than how it *should* be working.

These views provide another means of analyzing the execution of your application, helping to focus your optimization efforts. The difference between the Object and Class Interaction Views is that the Object Interaction View displays each individual instance of each class separately. An example of the Class Interaction View can be seen in Figure 17-21 on page 789.

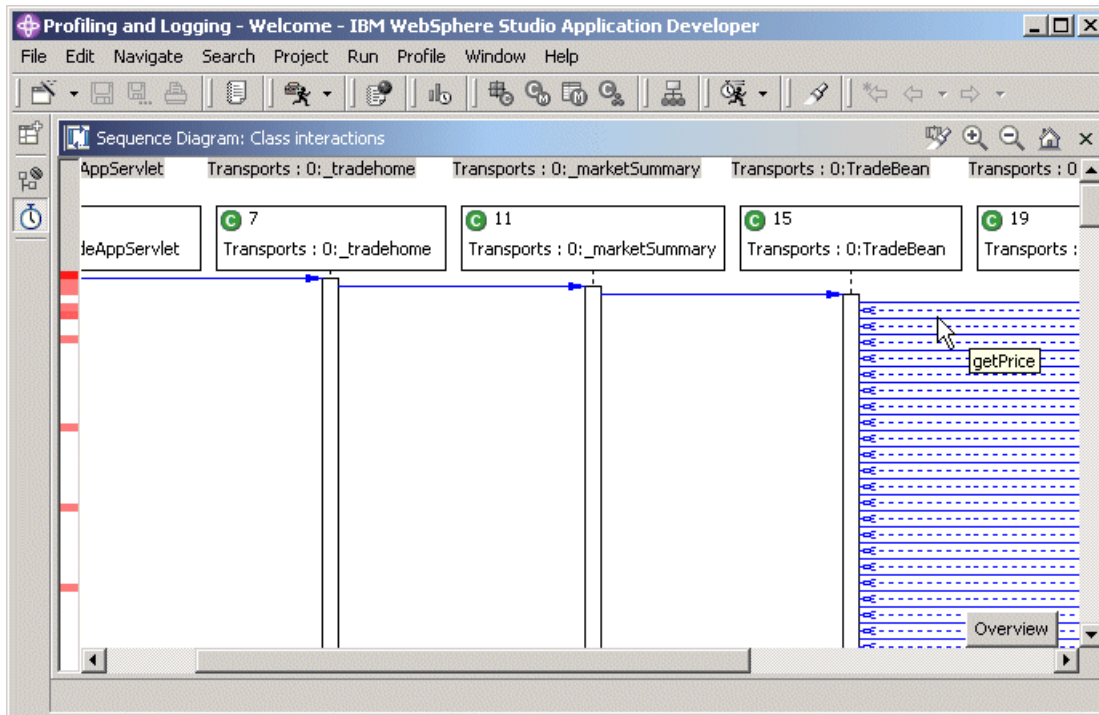


Figure 17-21 Class Interaction View

## Method Invocation View

The Method Invocation View is shown in Figure 17-22 on page 790. Using this view, it is possible to drill down to the methods shown in the Execution Flow View.

You can navigate to the Method Invocation View from the context menus of the Class Statistics Table or Method Statistics Table or Execution Flow View.

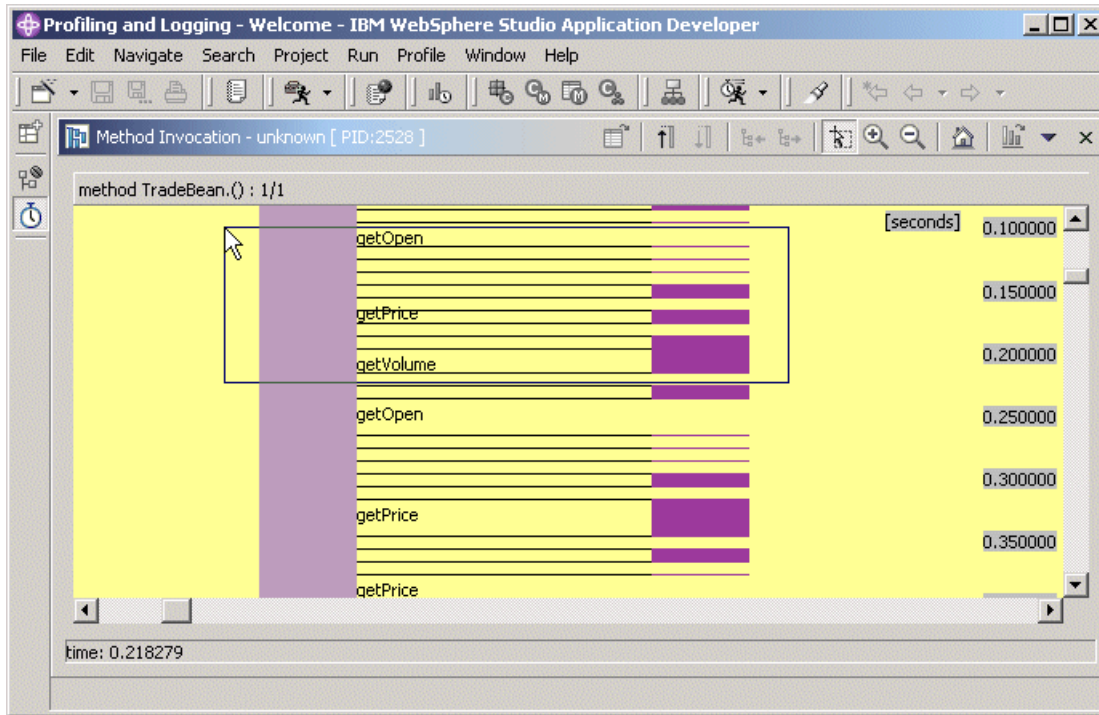


Figure 17-22 Method Invocation View

## Method Invocation Table

The Method Invocation Table contains the same information as the Execution Flow Table. This can only be accessed from the context menu of the Method Invocation View. This table is available only if the collection of execution flow information has been enabled.

## Host Interaction View

The Host Interaction View is the highest possible level of diagram available. Use this diagram to view interactions among methods that run on different hosts. It opens from the context menu of the monitor resource as shown in Figure 17-23 on page 791.



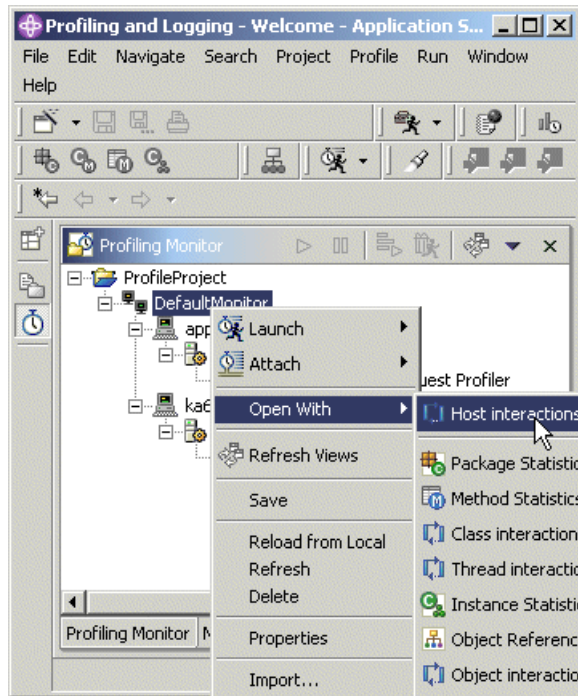


Figure 17-23 Open Host Interaction View

## Process Interaction View

Use the Process Interaction diagram to view the interactions among methods that run in different processes of the same host. This view opens from the context menu of the host as shown in Figure 17-24 on page 792.

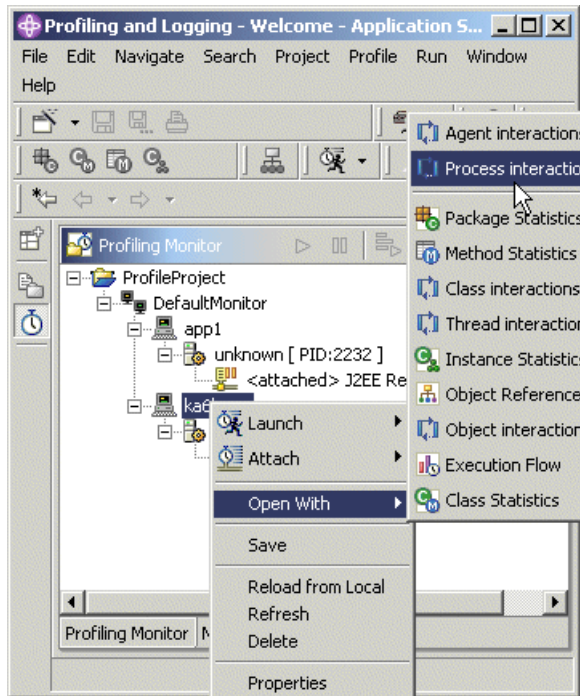


Figure 17-24 Open Process Interaction View

## Thread Interactions View

Use the Thread Interaction diagram to view the interactions among methods that execute in different threads of the same process. This view opens from the context menu of the resource that represents a monitor, host, or a process.

## 17.3 IBM Page Detailer

IBM Page Detailer is a browser side tool to measure performance of a Web application. While the Profiler discussed in the previous sections supports analysis of the execution of the application on the server, the Page Detailer collects most of its useful data at the socket level to reveal the performance details of items in the Web page, from the client's (browser's) perspective. It is also useful for measuring the incremental impact of changes in a Web application.

Page Detailer allows you to look at how and when each item is loaded in a Web page. Analyzing this data allows you to identify the areas where performance could be improved. The user's perception of performance is determined based



on the time to display pages, so measuring and analyzing this data will provide insight into the user's experience of your application.

### 17.3.1 Overview

IBM Page Detailer is a graphical tool that enables Web site developers and editors to rapidly and accurately assess performance from the client's perspective. IBM Page Detailer provides details about the manner in which Web pages are delivered to Web browsers. These details include the timing, size, and identity of each item in a page. This information can help Web developers, designers, site operators, and IT specialists to isolate problems and improve performance and user satisfaction. Page Detailer can be used with any site that your browser can access.

Page Detailer is a separately downloadable product that can be obtained from IBM alphaWorks® at:

<http://www.alphaworks.ibm.com/tech/pagedetailer>

There are two versions available:

- ▶ **Evaluation version:** This version is for free but does not support all features.
- ▶ **Pro version:** This version must be licensed for a small fee and contains the following additional features:
  - Full support for HTTPS (SSL) traffic
  - Saving and restoration of captured data
  - Ability to add/edit one's own notes for captured pages and items
  - A find facility for working with text

The supported platforms for Page Detailer are Windows 2000 and Windows XP.

The Page Detailer can monitor all HTTP and HTTPS requests originating from the machine where it is running. Thus it can be used to measure performance for Web applications running either locally or remotely, on either WebSphere Application Server or IBM WebSphere Studio Application Developer or any other Web site or application. Hence the Page Detailer may be used at any time during the project development cycle, from coding through to production support. Note that when analyzing the performance of non-production environments, differences in the production environment topology and configuration could result in differences in the measured performance results.

IBM Page Detailer gathers the following information:

- ▶ Connection time
- ▶ Socks connection time and size
- ▶ SSL connection time and size

- ▶ Server response time and size
- ▶ Content delivery time and size
- ▶ Delays between transfers
- ▶ Request headers
- ▶ Post data
- ▶ Reply headers
- ▶ Content data
- ▶ Page totals, averages, minimums, and maximums

For each page that is accessed, a color-coded bar chart of the time taken to load the page items will be generated. The length of a particular bar gives a good idea of the relative time spent in loading that item, as compared to the whole page. You will see that in some cases, items of a page may be loaded in parallel. This will appear in the chart with bars that overlap. The information that is captured by the Page Detailer includes page size as well as sizes of all other items loaded by the browser.

Different colors in the bar indicate how the time was spent.

- ▶ Page Time (Purple)  
The time taken to load all the components of a page.
- ▶ Host Name Resolution (Cyan)  
The time spent to resolve the IP address of the host.
- ▶ Connection Attempt Failed (Brown)  
The time taken to receive an error when a connection attempt is made.
- ▶ Connection Setup Time (Yellow)  
The time taken to open a socket connection. If a SOCKS server is being used, this is the time to open a socket connection from the browser to the SOCKS server only.
- ▶ Socks Connection Time (Red)  
The time taken to open a connection from a SOCKS server to the remote site.
- ▶ SSL Connection Setup Time (Pink)  
This is the time taken to negotiate an encrypted connection between the browser and the remote site, once a normal socket connection has been established.
- ▶ Server Response Time (Blue)  
This is the time from the browser's request to the receipt of the initial reply, after all the communications setup has been completed. Large responses are broken down into smaller components (packets). The server response time only measures the time to receive the first one.

► Delivery Time (Green)

The time taken to receive all additional data that was not included in the initial response.

An example of a chart produced with Page Detailer is provided in Figure 17-25.

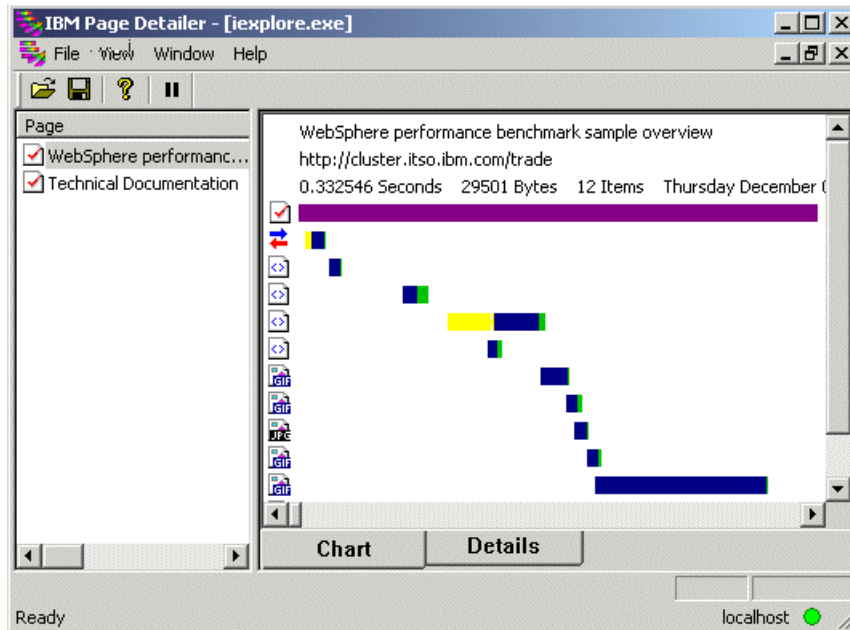


Figure 17-25 Page Detailer Chart View

To obtain more detailed information about a particular HTTP request, double-click the appropriate colored bar or the icon in the chart. This will display a text viewer as shown in Figure 17-26 on page 796. It includes information about timings, sizes, header, etc. You can add your own notes and save the file for comparison. This feature is available only in the Pro version.

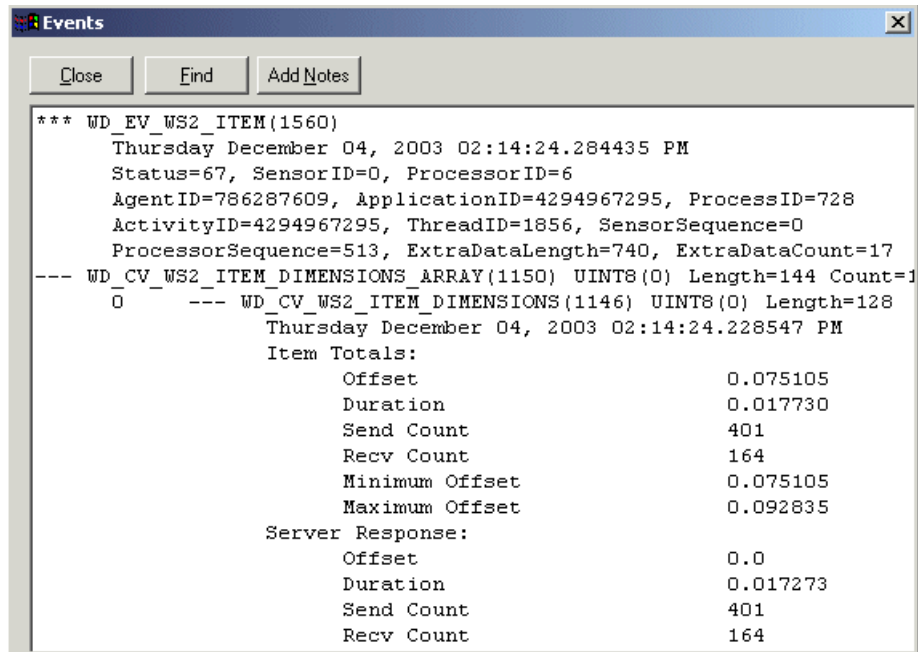


Figure 17-26 Page Detailer Events

In addition to the Chart view, it is also possible to view more details of all HTTP requests using the Details view. This can be seen by selecting the **Details** tab at the bottom of the window (see Figure 17-27 on page 797).

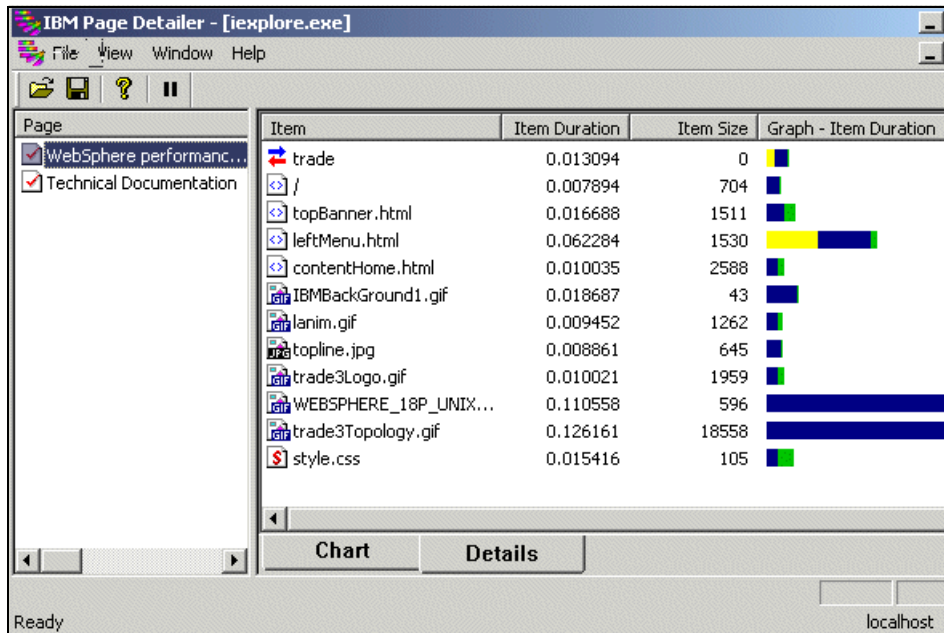


Figure 17-27 Page Detailer Details View

## 17.3.2 Important considerations

Some important considerations while taking measurements are as below:

- Impact of network delays

Many problems may not be evident when accessing a server on a local network, but may become apparent when accessing the site remotely, particularly when using a modem connection. On the other hand, you can minimize the effect of external network delays by directly connecting on the Server's LAN. This will allow you to isolate the performance impact of a change made to the Web page.

- Browser cache

Disabling browser cache helps in getting repeatable results. However you could also check the performance from a user's perspective by enabling the browser cache and comparing both results.

- Packet loss

Packet loss can happen and get corrected in the underlying TCP/IP layers. This is invisible to the Page Detailer. Packet loss manifests itself as inconsistent time measurements in Page Detailer. You can take a series of measurements at different times to factor it out.

### 17.3.3 Key factors

Some of the key factors that influence the time to load a Web page in a browser are:

- ▶ Page size.
- ▶ Number, complexity, and size of items embedded in the page.
- ▶ Number of servers that need to be accessed to retrieve all elements, and their location and network connectivity.
- ▶ Use of SSL (this introduces an extra overhead).

The Page Detailer will help you to identify when one of these problems is affecting some or all of your application. It will also help to identify problems such as broken links and server timeouts.

Some of the strategies that can be used to improve performance and resolve problems you have identified include:

- ▶ Minimize the number of embedded objects. Avoid the excessive use of images in particular. In cases where there is a standard header, footer, or side menu on every screen, consider the use of frames so that common elements do not have to be downloaded every time.
- ▶ The browser will typically retrieve multiple items in parallel, in the order in which they appear in the HTML page that it receives. Hence sequencing of the items so that downloads for larger objects are started early can reduce the total time required to display the page, and avoid the user having to wait for a long time for the last element(s) to be retrieved.
- ▶ Ensure that caching is being used effectively. Often the same images are used multiple times on the same page. If there are two references to the same image in close proximity to each other in the HTML source, the browser may encounter the second reference before the HTTP request that was initiated to download the first reference has been completed. In this case the browser may issue another request to retrieve the image again. This can be avoided by pre-loading frequently used images multiple times early, or by structuring the generated pages so that such URLs do not appear consecutively.
- ▶ Minimize the use of SSL where possible. For example, some content such as images may not need to be secured even though the application as a whole needs to be secure.
- ▶ Try to avoid switching the user to an alias server name during the page load. This will help the browser to reduce the look up time and possibly avoid a new connection.

### 17.3.4 Tips for using Page Dettailer

Here are a few helpful tips to analyze the performance data shown by Page Dettailer. A sample analysis, with comments, is shown in Figure 17-28.

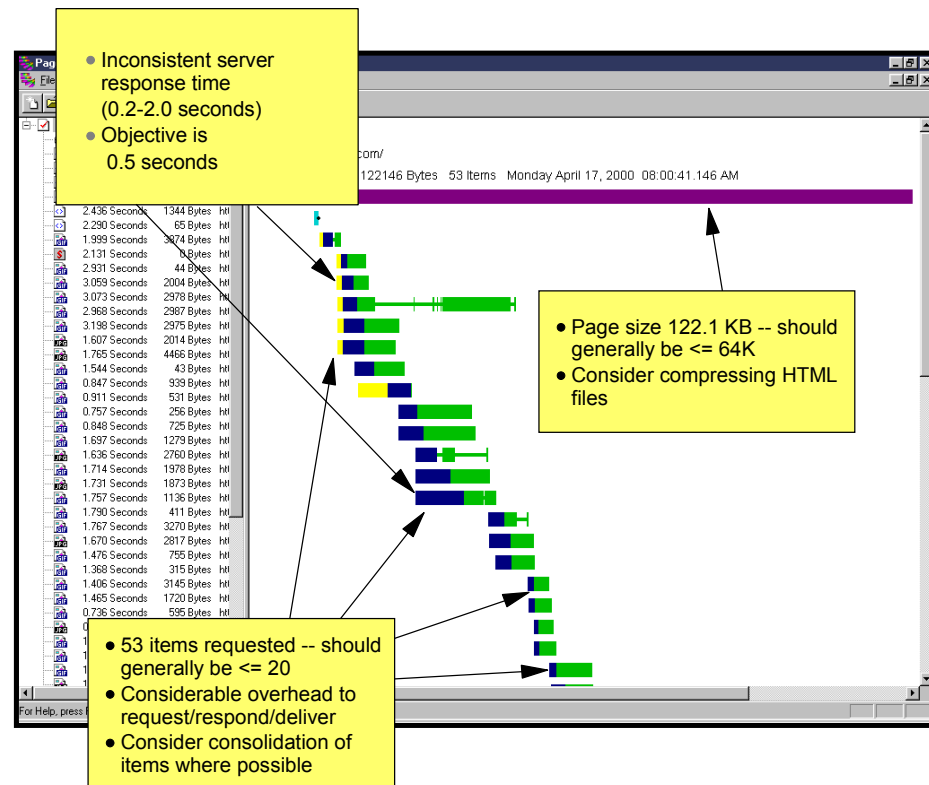


Figure 17-28 Page Dettailer Sample Analysis

- A summary of above information and the meaning of each icon and color can be obtained by using the menu **View -> Legend**. This displays the window shown in Figure 17-29 on page 800.

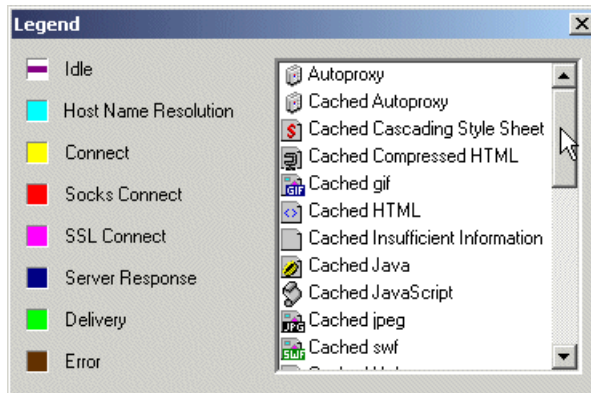


Figure 17-29 Page Detailer Legend

- ▶ Use a separate browser instance to collect data from related Web pages in one file. This allows for easy retrieval of performance data of related Web pages for comparison. Please note that you can save your work only in the Pro version of the Page Detailer.
- ▶ You can select the columns to display in the Details view from the context menu (obtained by right-clicking). You can also choose to display the column as a graph if it contains a numerical value.
- ▶ You can move the vertical bar and make room to add new columns on the left hand side of the window as shown in the Figure 17-30 on page 801 and Figure 17-31 on page 802.



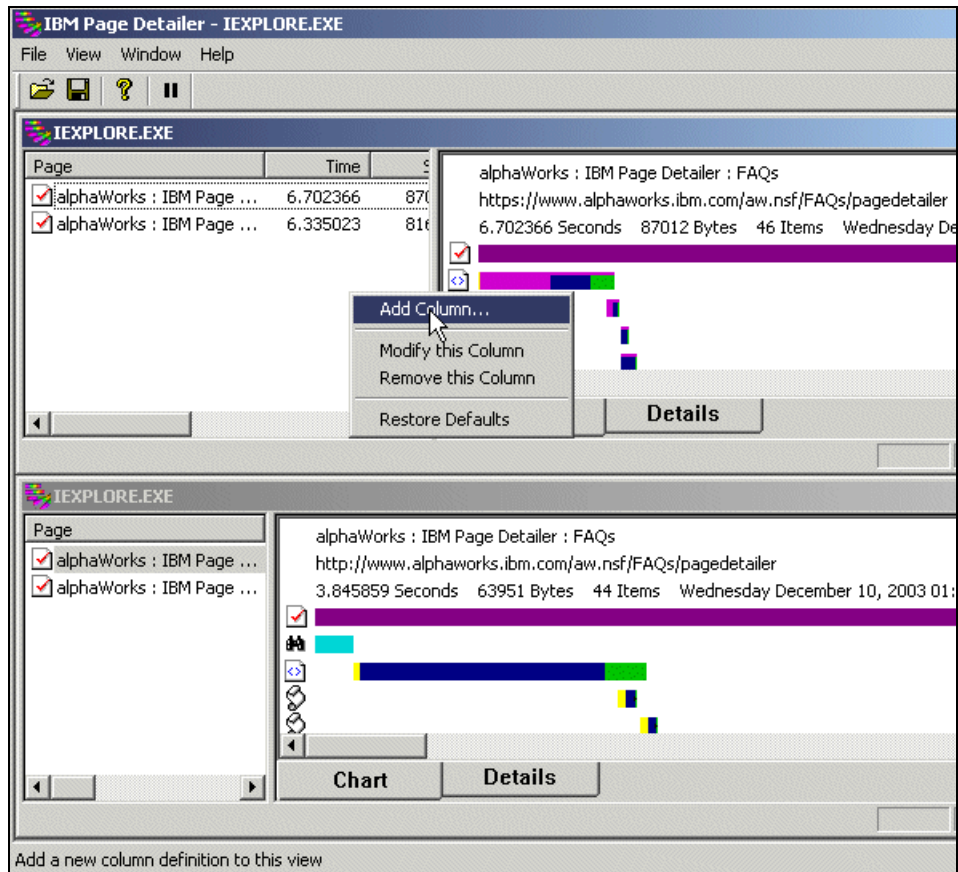


Figure 17-30 Moving the separator and adding a new column

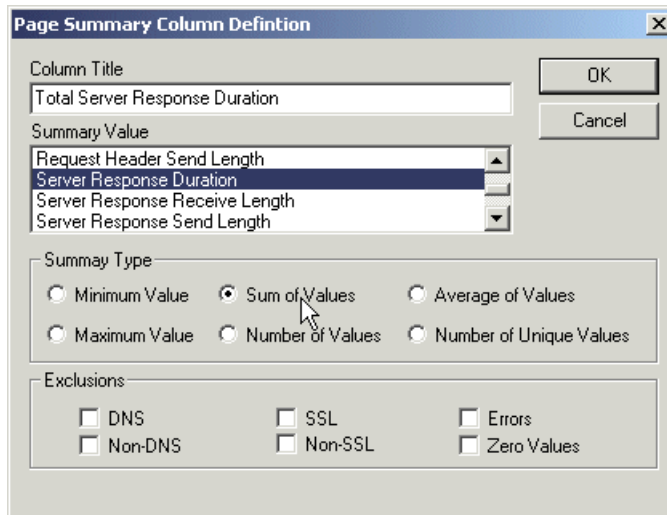


Figure 17-31 Column Definition

### 17.3.5 Reference

- ▶ See the article “Design for Performance: Analysis of Download Times for Page Elements Suggests Ways to Optimize” at:  
<http://www.ibm.com/developerworks/websphere/library/techarticles/hvws/perform.html>
- ▶ See the Page Detailer Web site at IBM alphaWorks:  
<http://www.alphaworks.ibm.com/tech/pagedetailer>



# **Application development: Best practices for performance and scalability**

This chapter discusses best practices for developing WebSphere Application Server-based applications.

## 18.1 Introduction and general comments

This chapter discusses coding best practices for developing WebSphere Application Server based applications that perform well and are scalable and robust. Best practices for general Java coding as well as specific guidelines for Servlets, JavaServer Pages (JSPs), Enterprise JavaBeans (EJBs) and the use of J2EE resources are discussed.

However, a prerequisite is to ensure that your application has a good design and architecture. The techniques and strategies outlined here will assist you in optimizing the performance of your applications. However, they will not compensate for a poorly designed or architected application. The best practices should be applied after verifying that the basic design and architecture are appropriate for the application and are scalable.

It is also important to reuse existing patterns and algorithms where appropriate, rather than “reinventing the wheel”. There are established algorithms and approaches for solving many of the commonly encountered problems in computer science, and any decision to use a custom solution in preference to one of these commonly used solutions should be made with caution.

Inefficient code is written for a number of reasons. These include a lack of focus on performance, a limited awareness of performance issues by the developers, or a desire to produce an elegant or highly object-oriented solution. It is important that performance targets are set at the beginning of the project and communicated to all team members involved in development. The implications of these targets on development work should also be analyzed early in the development cycle and made known.

Please note that often there is a trade-off between implementations that are elegant, object oriented, easy to read and maintain, and those that offer higher performance. In these cases, the use of profiling tools such as those included with IBM WebSphere Studio Application Developer described in Chapter 17, “Development-side performance and analysis tools” on page 759 can be extremely useful. These tools will indicate which parts of the code are most frequently used, and also where the majority of the execution time is spent. Typically, the majority of the execution time will be spent in a minority of the code, as suggested by the 80/20 rule - 80% of the execution time will be spent in 20% of the code. In many cases, the ratio may be even higher, such as 90/10 or more. Although it is important to be aware of best practices when performing all development work, extra care should be taken to optimize the most frequently used code sections. More information on server-side tools for analyzing usage patterns of components of an application such as servlets and EJBs can be found in Chapter 16, “Server-side performance and analysis tools” on page 685.

In order to optimize performance experienced by the end users, ongoing tuning activities should be undertaken throughout the life cycle. Monitoring of the application during testing can indicate performance problems that are not apparent in the development environment. Additionally, specific testing of performance under simulated loads using a tool such as OpenSTA, Apache JMeter, Mercury LoadRunner, or Rational Robot will also provide insight to where performance optimization work should be targeted. Performance tuning is an iterative process that continues once the coding phase of a project is complete, through the life cycle of the project. Developing a prototype early in the development cycle can provide valuable insight into the expected performance, and will help to ensure that the implementation approach is appropriate and will perform adequately.

**Note:** For more information about stress testing an application, refer to 19.1, “Testing the performance of an application” on page 822.

## 18.2 Memory

A key factor in the performance of any Java application and hence any WebSphere Application Server application is the use of memory. Unlike other programming languages, Java does not require (or even allow) programmers to explicitly allocate and reclaim memory. The Java Virtual Machine (JVM) runtime environment will allocate memory when a new object is created, and will reclaim the memory once there are no more references to the object. This reduces the amount of coding required, as well as minimizing the potential for memory “leaks” caused by the programmer forgetting to deallocate memory once it is no longer required. Additionally, Java does not allow pointer arithmetic. Memory deallocation is performed by a thread executing in the JVM called the *garbage collector* (GC). The algorithm used for garbage collection is not specified in the Java Virtual Machine specification and hence different JVM implementations may use different garbage collection algorithms.

However, all of these algorithms require the sweeping of memory to identify objects that are no longer referenced. Most also involve a compaction phase where referenced objects are copied to a particular area of memory to reduce fragmentation. More details about garbage collection can be found in “Improving Java Application Performance and Scalability by Reducing Garbage Collection Times and Sizing Memory” by Nagendra Nagarajayya and J. Steven Mayer which is available at:

<http://developers.sun.com/techtips/mobility/midp/articles/garbage/>

An update to this article for JDK 1.4.1 is also available, it is found at:

<http://developers.sun.com/techtips/mobility/midp/articles/garbagecollection2/>

Although Java performs memory management automatically, programmers still need to be aware of the impact of memory management on performance. Creating an object consumes system resources, because the JVM must allocate memory and initialize the object. Similarly reclaiming memory using the garbage collector also uses resources, particularly CPU time. Garbage collection occurs asynchronously when free memory reaches threshold values, and it cannot be explicitly scheduled programmatically. A call to the `System.gc()` method will request that the JVM performs garbage collection. However, this is not guaranteed to happen immediately or within any specified time period.

Hence the key to minimizing the performance impact of memory management is to minimize memory usage, particularly object creation and destruction. This can be achieved by a number of means:

- ▶ Object creation

Don't create objects prematurely if there is a possibility that they will not be needed. For example, if the object is only used in one path of an if statement, then create the object inside that path rather than outside the if statement - lazy initialization. If the same object can be reused inside a loop body, then declare and instantiate it outside the loop rather than inside the loop, to avoid creating and destroying a number of objects of the same class.

- ▶ Object pools

If objects of the same class are being repeatedly created and destroyed, it can be beneficial to create an object pool that allows the objects to be reused. Classes whose objects will be used in a pool need an initializer, so that objects obtained from the pool have some known initial state. It is also important to create a well-defined interface to the pool to allow control over how it is used.

**Note:** IBM WebSphere Application Server Enterprise V5.1 provides Object Pools for pooling application defined objects or basic JDK types. It will benefit an application which tries to squeeze every ounce of performance gain out of the system.

- ▶ Appropriate sizing for collections

Although the Java runtime environment will dynamically grow the size of collections such as `java.util.Vector` or `Java.util.Hashtable`, it is more efficient if they are appropriately sized when created. Each time the collection size is increased, its size is doubled so when the collection reaches a stable size it is

likely that its actual size will be significantly greater than required. The collection only contains references to objects rather than the objects themselves, which minimizes the overallocation of memory due to this behavior.

- Temporary objects

Developers should be aware that some methods such as `toString()` methods can typically create a large number of temporary objects. Many of the objects may be created in code that you don't write yourself, such as library code that is called by the application.

- Use of static and final variables

When a value is used repeatedly and is known at compile time, it should be declared with the `static` and `final` modifiers. This will ensure that it will be substituted for the actual value by the compiler. If a value is used repeatedly but can be determined only at runtime, it can be declared as `static` and referenced elsewhere to ensure that only one object is created. Note that the scope of static variables is limited to the JVM. Hence if the application is cloned, care needs to be taken to ensure that static variables used in this way are initialized to the same value in each JVM. A good way of achieving this is the use of a *singleton* object. For example, an EJB initial context can be cached with a singleton (as suggested in 18.7, "Enterprise JavaBeans" on page 813) using the following code fragment:

*Example 18-1 Use of the singleton pattern to cache EJB initial context references*

---

```
public class EJBHelper {
    private static javax.naming.InitialContext initialContext= null;

    public javax.naming.InitialContext getInitialContext() {
        if (initialContext == null) {
            initialContext = new javax.naming.InitialContext();

            return initial Context
        }
    }
}
```

---

- Object references

Although memory does not have to be explicitly deallocated, it is still possible to effectively have "memory leaks" due to references to objects being retained even though they are no longer required. These objects are commonly referred to as *loitering objects*. Object references should be cleared once they are no longer required, rather than waiting for the reference to be implicitly removed when the variable is out of scope. This allows objects to be reclaimed sooner. Care should be taken with objects in a collection, particularly if the collection is being used as a type of cache. In this case,

some criteria for removing objects from the cache is required to avoid the memory usage constantly growing. Another common source of memory leaks in Java is due to programmers not closing resources such as Java Database Connectivity (JDBC), Java Message Service (JMS) and Java Connector Architecture (JCA) resources when they are no longer required, particularly under error conditions. More information about the use of JDBC resources is provided in 18.8, “Database access” on page 816. It is also important that static references be explicitly cleared when no longer required, because static fields will never go out of scope. Since WebSphere Application Server applications typically run for a long time, even a small memory leak can cause the Java Virtual Machine to run out of free memory. An object that is referenced but no longer required may in turn refer to other objects, so that a single object reference can result in a large tree of objects which cannot be reclaimed. The profiling tools available in IBM WebSphere Studio Application Developer V5.1.1, which are described in Chapter 17, “Development-side performance and analysis tools” on page 759, can help to identify memory leaks. Other tools that can be used for this purpose include Rational Purify®, Sitraka JProbe (by Quest Software), and Borland Optimizelt.

- Vertical clustering

Most current garbage collection implementations are partially single threaded (during the heap compaction phase). This causes all other program threads to stop, potentially increasing the response times experienced by users of the application. The length of each garbage collection call is dependent on numerous factors, including the heap size and number of objects in the heap. Thus as the heap grows larger, garbage collection times can increase, potentially causing erratic response times depending on whether a garbage collection occurred during a particular interaction with the server. The effect of this can be reduced by using vertical scaling and running multiple copies of the application on the same hardware. This can provide two benefits: first, the JVM for each member of the cluster will only require a smaller heap, and secondly, it is likely that while one JVM is performing garbage collection, the other one will be able to service client requests as the garbage collection cycles of the JVMs are not synchronized in any way. However, any client requests that have been directed by workload management to the JVM (doing garbage collection) will be affected. Refer to 3.7, “Vertical scaling” on page 67 for more information on vertical clustering.

## 18.3 Session Management

In a Web application, state information relating to each client is typically stored in an HTTP session, which is identified by some unique identifier that is associated with an HTTP cookie. In an environment with a single application server, session information can be stored in-memory by WebSphere Application Server V5.1.



However, it is more common to use a clustered environment with multiple application servers to provide scalability and improve fault tolerance.

In this scenario, session information needs to be made available across all of the cluster members. In past versions of WebSphere Application Server, this was achieved using a session persistence database that was available to all clones in a server group. In addition to this, a new mechanism for memory-to-memory replication has been introduced in IBM WebSphere Application Server V5.1. Refer to “Managing session state among servers” on page 21 and to “Session persistence considerations” on page 59 for more information about session management, database session persistence and memory-to-memory replication.

In either case, it is beneficial to minimize the amount of data stored in the session. Since the session must be shared, it must be serialized, which also involves serializing all objects that are reachable from the session. Serialization in Java is an expensive operation. If persistent sessions are used, the serialized session data must be stored in the database, which introduces further overhead. Usually the session is stored as a binary data type (BLOB) in the database, which also may introduce overhead.

In order to reduce the amount of data stored in the session, avoid storing large, complex object graphs in it. In some cases it may be beneficial to store objects in the session, although they can be recreated or retrieved to avoid the overhead of doing so. In these cases, consideration should be given to making these attributes transient. If this is done, you have to ensure that the application code will handle the transient attributes having null values. Alternatively, the `readObject()` method of the object could be overwritten to recreate the transient data when the object is deserialized.

The session object can be garbage collected after it has been invalidated. This can be done programmatically or after a predefined timeout period during which the session was not accessed. To allow the memory used by the session to be reclaimed as early as possible, it is best to explicitly invalidate the session when finished with it rather than waiting for the timeout. This may require the introduction of logout functionality into the application, and training for the users to make use of this functionality rather than simply closing the browser.

References to the session should always be obtained from the current servlet context as required - they should not be cached by the application. This ensures that the session objects can be reclaimed when the session is invalidated. Another reason for doing this is because a session reference is not guaranteed to be valid outside of the context of a specific server interaction.

Special care must be taken when using HTML frames when each frame is displaying a JSP belonging to a different Web application on the same server. In this case, a session should only be created and accessed by one of the pages.

Otherwise, although a session will be created for each page, the same cookie will be used to identify the session. This means that the cookie for each newly created session will overwrite the previous cookie, and only one of the sessions will be accessible. The remaining sessions will be created but will be inaccessible and thus will consume memory until the timeout interval is reached. If the Web application was split into multiple applications in order to improve scalability, consider combining all of the Web applications into a single one, and using clustering to achieve the required scalability.

There are alternatives to the use of the session that may be appropriate in some situations:

- ▶ In some cases use of the session can be avoided by using hidden form fields or cookies to store data. Note that there is a 4 KB limit on the total size of all cookies for a particular site. Also be aware that the use of hidden fields increases the page size and the data can be seen by the user when viewing the HTML source.
- ▶ Data can be stored directly in a database by the application. By using native data types instead of serialized BLOBs, it is often possible to achieve better performance. It is also possible to read and write only the data that has changed, rather than the entire data set as is normally the case with BLOBs. The application will need to remove data when it is no longer required (including after a timeout period). This can be implemented by placing an object that implements the `HttpSessionBindingListener` interface into the session, and placing the clean up code in the `valueUnBound()` method.
- ▶ Entity EJBs can be used to store session data. However the performance implications of this approach need to be assessed, before implementing this type of solution. In particular, this may not perform well if complex data structures need to be stored. For more information on best practices for EJBs, refer to “Enterprise JavaBeans” on page 813.

## 18.4 Synchronization

The mechanism by which access to shared resources by different threads is controlled is called *Synchronization*. While the synchronization functionality in Java is convenient and easy to use, it can introduce significant performance overhead. When a block of code is synchronized, only a single thread can execute it at any one time. There are two performance impacts of synchronization:

- ▶ Managing the *monitors*, the objects internal to the JVM that are used to control access to synchronized resources. Although they are not explicitly accessed by programmers, there is an overhead due to the management of the monitors by the JVM.

- ▶ Reduced concurrency, since threads have to wait for other threads to exit from synchronized blocks of code.

Thus the use of synchronization should be minimized and limited to cases where it is definitely necessary. It is also good practice to clearly document all assumptions relating to synchronization, because they may not be obvious to someone reading the design or code.

When using synchronization, it is best to use specific lock objects to synchronize on. Synchronizing using the keyword can cause different methods to be unnecessarily synchronized with each other, and hence reduce concurrency. Note that synchronizing on an object has a greater overhead than calling a synchronized method. However, synchronizing the method may result in significantly greater amounts of code being synchronized, again reducing the concurrency. So the trade-off between the synchronization overhead and reduced concurrency needs to be evaluated on a case-by-case basis.

In addition to the explicit use of synchronization in application code, synchronization may be used indirectly, as some of the commonly used core Java functionality uses synchronization. Some particular examples are:

- ▶ The Java I/O libraries. It is best to minimize the use of `System.out.println()` for this reason. Use of a multithreaded logging library as discussed in 18.6, “Logging” on page 812 is suggested.
- ▶ Some of the Java collection classes, such as `java.util.Hashtable` and `java.util.Vector`, are synchronized. If only a single thread is accessing the data (or multiple threads are reading only), the synchronization overhead is unnecessary. Many of the newer collections introduced in Java 1.2, such as `java.util.ArrayList` are not synchronized and may provide better performance. However, care needs to be taken when accessing them from multiple threads.

## 18.5 Servlets and JavaServer Pages

Since servlets and JavaServer Pages (JSPs) can include Java code, many of the issues discussed in other sections of this chapter are relevant for JSPs as well. However, there are some particular issues that need to be considered when developing JSPs:

- ▶ Minimize the use of the `<jsp:include>` tag, since each included JSP is a separate servlet.
- ▶ When a `<jsp:usebean>` tag is encountered and an existing Java bean object with the appropriate name does not exist, a new one is created. This is done by a call to `Beans.instantiate()`, which is an expensive operation because the JVM checks the file system for a serialized bean. Hence it is recommended

that the `<jsp:usebean>` tag only be used to obtain a reference to an existing object, rather than for creating a new object.

- ▶ In accordance with the Java 2 Enterprise Edition (J2EE) specification, when executing a JSP, a session object is normally created implicitly if one does not already exist. However if the session is not required, creation can be avoided by the use of the `<%@ page session="false" %>` directive.
- ▶ Avoid the use of the `SingleThreadModel` for servlets, since it permits only one servlet thread to be executing at any time. Other client requests will be queued, waiting until a servlet to process them becomes available. This reduced concurrency can significantly reduce the throughput, and consequently increase the response times experienced by users. If a servlet has shared variables that need to be protected, it is preferable to do so using synchronization of the relevant accesses. The `SingleThreadModel` is effectively equivalent to synchronizing the servlets entire `service()` method.
- ▶ The `javax.servlet.Servlet.init()` method can be used to perform expensive operations that need to be performed once only, rather than using the `doGet()` or `doPost()` methods of the servlet.

## 18.6 Logging

As mentioned in “Synchronization” on page 810, the Java I/O classes use synchronization. Hence `System.out.println()` should not be used for logging purposes. If a lot of output using `stdout` is generated by an application in a UNIX environment, the overhead can be avoided by redirecting `stdout` to `/dev/null` in the production environment. However, a better approach is to use a multithreaded logging library such as the WebSphere logging facilities or `Log4J` (<http://jakarta.apache.org/log4j/>). In addition to providing better performance due to their multithreaded implementations, these libraries allow logging statements to be defined at a particular level, which can be dynamically changed at runtime. Thus the amount of logging in production environments can be reduced in comparison to development and test environments without requiring code changes, improving the performance of the application. It is also good practice to guard log statements so that the parameters are not evaluated if the logging level is not on. The use of guard statements is shown in Example 18-2.

*Example 18-2 Use of guard statements for logging*

---

```
if (Log.isLogging(Log.WARN) {  
    Log.log(LOG.WARN, "This is a warning");  
}
```

---

## 18.7 Enterprise JavaBeans

Enterprise JavaBeans (EJBs) are an important aspect of the J2EE specification, and appropriate use of EJBs can support the development of modular and scalable applications. There are, however, a number of performance considerations that need to be taken into account when using EJBs:

- Obtaining EJB references

Since EJBs are able to be accessed remotely, obtaining a reference to an EJB involves a lookup process. This can take a relatively long time, and hence caching the references obtained can improve performance on subsequent lookup operations. Obtaining an EJB reference typically involves the following steps:

- a. Obtain an InitialContext instance.
- b. Obtain a reference to the home interface of a particular EJB by looking up the EJB via the initial context object and performing a narrow operation.
- c. Obtain a particular EJB instance by executing the appropriate finder or create method from the EJB home instance.

The calls to obtain an InitialContext instance and lookup of an EJB Home instance (steps a and b) are expensive, and performance can be improved by caching these objects. References to EJB homes may become stale, and hence it is best to wrap accesses to EJB Home methods with code that refreshes stale references.

- Remote method calls

Since EJBs are intended to be accessible remotely, calls to EJB methods are implemented as remote calls even if the EJB exists in a container that shares the same JVM as the Web container, which introduces some overhead. In some cases, the overhead can be reduced by implementing approaches outlined below.

One of the key mechanisms to reduce EJB overhead is to minimize the number of remote calls. In general, a servlet should make a single EJB method call to a session bean to perform an operation. If required, this EJB method can call other methods as required to perform more complex operations. All access to Entity EJBs should be performed through stateless session beans. Session beans should be used to implement the business logic, while entity beans should be used for data storage and retrieval.

IBM WebSphere Application Server V5.1 is compliant with the J2EE 1.3 specifications, which mandate support for EJB 2.0. The concept of a local interface has been introduced for EJB 2.0. If local interfaces are used, the overhead of serializing the parameters and transmitting them via RMI/IIOP (Remote Method Invocation/Internet Inter-ORB Protocol) is avoided.

Parameters can be passed by reference instead of value, avoiding the need to copy them. In EJB 2.0, an EJB can have a remote interface, local interface or both, although typically a particular bean will have either a remote or a local interface. Since Entity EJBs are normally accessed from session beans, in most cases they will only need local interfaces.

► Transaction management

In addition to the overhead associated with remote method calls, transaction management provided by the EJB container can also introduce overhead. Accessing entity beans from session beans can limit the impact of this by reducing the number of transactions. In the deployment descriptor for EJBs, the transaction type can be specified. This can be one of five values: NotSupported, Supports, Required, Requires New and Mandatory. Set the transaction type to NotSupported if no transaction is required.

► Entity EJBs

While entity beans can reduce the amount of coding work required by freeing programmers from having to write code for persisting data, care must be taken with the use of entity beans to avoid performance problems. As mentioned earlier, it is recommended that entity beans are only accessed from session beans and not directly by servlets or other clients. The EJB access beans functionality provided in IBM WebSphere Studio Application Developer V5.1.1 can also be used to simplify access to EJBs.

In cases where entity beans have deep relationships or significant use of inheritance, care must be taken to ensure that performance is adequate. If performance problems are encountered, they can potentially be addressed by reducing the use of inheritance in the model or by use of bean-managed persistence (BMP) in preference to container-managed persistence (CMP). Another strategy is to avoid turning each entity (table) into a single EJB - in some cases two or more related entities can be represented by a single EJB.

Entity EJBs are best used for manipulating small amounts of data. Returning large result sets from (default) EJB finders can be inefficient and it is often better to use JDBC directly from a session bean for this.

**Important:** The following applies only to EJBs that are compliant with the EJB 1.1 specification. We recommend that new EJBs be developed to be compliant with the EJB 2.0 specification. However, some existing EJBs may only be compliant with EJB 1.1.

– Isolation levels and read-only methods

If an entity bean has methods that do not update attributes (getter type methods) they can be specified as read-only methods in the deployment

description. This will avoid executing a SELECT for UPDATE/UPDATE pair of SQL statements, and hence will provide a performance improvement.

EJBs deployed in WebSphere Application Server can be assigned one of four transaction isolation levels. These are Repeatable Read, Read Committed, Read Uncommitted and Serializable. The default level is Repeatable Read, but performance can be improved by the use of a lower isolation level such as Read Committed.

**Important:** The following applies only to EJBs that are compliant with the EJB 2.0 specification.

- Access intent

The access intent for each method of an EJB 2.0 CMP entity bean can be specified in the EJB deployment descriptor to provide hints to the EJB container about how the method will be used. The supported access intents are pessimistic update - Weakest Lock At Load, pessimistic update, optimistic update, pessimistic read, and optimistic read. In general, read-only methods and optimistic locking will provide better performance. However, be careful with the use of optimistic locking since problems may only become apparent under heavy load and hence may not be found in development.

- Stateful session beans

The use of stateful session beans should be minimized (or avoided if possible). Unlike stateless session beans, they are not able to be work load managed or pooled. The EJB container may passivate inactive stateful session beans and activate them if they are referenced later. This involves serializing and deserializing the beans, which can impact the performance. In cases where stateful session beans are required, the occurrence of passivating the beans can be reduced by explicitly removing the beans when they are no longer required, rather than waiting for them to time out.

One scenario in which stateful session beans may be useful is the implementation of a cache for application data. This may use JDBC or a BMP entity bean to retrieve the data to be cached.

- EJB caching

The EJB container has three types of caching that can be performed for entity beans between transactions. Selection of the appropriate option requires an understanding of how the entity beans will be used, as there is a trade-off

between minimizing database reads and supporting Workload Management (WLM).

- Option A caching

With this option, it is assumed that the data represented by the EJB is exclusively accessed by the EJB container and is not accessed by other containers or by some mechanism that is external to the application server. Use of this option can improve performance, since the data does not have to be retrieved again from the database prior to each transaction. However if option A caching is used, the entity bean cannot be clustered or participate in WLM.

- Option B caching

The entity bean will remain in the cache throughout the transaction, but is reloaded at the start of each method call. This introduces the greatest overhead.

- Option C caching

The entity bean will be reloaded at the beginning of each transaction. This is the default setting, and allows for a client to access a particular bean from any container.

- ▶ Asynchronous message processing

Asynchronous Java Message Service (JMS) messages can be processed using message-driven beans (MDBs), which have been introduced in WebSphere Application Server V5.0. This avoids the need for developing individual server programs to receive and process incoming messages.

## 18.8 Database access

The Java Database Connectivity (JDBC) API provides a vendor-independent mechanism to access relational databases from Java. However, obtaining and closing a connection to a database can be a relatively expensive exercise, so the concept of *connection pools* has been introduced. When a database operation is to be performed, a connection can be obtained from the pool, which contains a defined number of connections to the database that have already been established. When the connection is closed, it is returned to the pool and made available for reuse. Using connection pooling can significantly reduce the overhead of obtaining a database connection. However, the connection pool is accessed via a data source. References to the data source are obtained by performing a lookup via the Java Naming and Directory Interface (JNDI). This lookup is an expensive operation, so it is good practice to perform the lookup once and cache the result for reuse.



JDBC resources should always be released once they are no longer required. This includes `java.sql.ResultSet`, `java.sql.Statement` and `java.sql.Connection` objects, which should be closed in that order. The code to close the resources should be placed in a `finally` block to ensure that it is executed even when an exception condition occurs. Failure to properly close resources can cause memory leaks, and can cause slow response due to threads having to wait for a connection to become available from the pool. Since database connections in the pool are a limited resource, they should be returned to the pool once they are no longer required.

If an application repeatedly executes the same query, but with different input parameters, then performance can be improved by using a `java.sql.PreparedStatement` instead of `java.sql.Statement`.

Turning off `autocommit` for read only operations may increase performance.

To avoid having to retrieve and process large amounts of data, sometimes it is beneficial to use database stored procedures for implementing some of the application logic. Alternatively, in some cases calls to the database can be minimized by using a single statement that returns multiple result sets.

There are different types of JDBC drivers available, some written in pure Java and others that are native. Although use of a native driver can reduce portability, performance may be better with a native driver.

## 18.9 General coding issues

This section describes a variety of techniques to improve performance of WebSphere applications, particularly through the efficient use of the core Java functionality.

- ▶ Although strings are a simple and efficient data structure in many languages such as C/C++, there is overhead associated with the use of strings (`java.lang.String`) in Java. Java strings are immutable; once created their value cannot be changed. Hence operations such as string concatenation (+) involve the creation of new strings with the data copied from the original strings, creating more work for the garbage collector as well. When performing string manipulation operations, use of `java.lang.StringBuffer` as an alternative to `java.lang.String` can improve performance.
- ▶ Although the reflection facilities in Java can be extremely useful and allow for elegant implementations, reflection is an expensive operation that should not be used indiscriminately. This is another case of where a trade-off between performance and elegance of the solution needs to be made.

- ▶ Avoid creating excessively complicated class structures. There is a performance overhead in loading and instantiating these classes.
- ▶ Avoid excessive and repeated casting. Once an object has been cast, assign a variable of the correct type and reuse this reference.
- ▶ Use of “?:” for a conditional check instead of if-then provides better performance for most JVMs.
- ▶ When iterating n items, iterating from n-1 to 0 instead of 1 to n is quicker for most JVMs. See Example 18-3.

*Example 18-3 Iterating through a loop n times*

---

```
for (int=n-1;i>0;i--) {
    // Do something in a loop....
}
```

---

- ▶ Avoid repeatedly calling the same method within a loop if the result is the same every time. Instead store the value in a variable prior to entering the loop and use this stored value for each loop iteration.
- ▶ Use `System.arraycopy()` to copy the contents of one array to another, rather than iterating across each array element and copying it individually.
- ▶ Where possible, declare methods with the `final` modifier if they will not be overridden. Final methods can be optimized by the compiler by method in-lining. The byte code of the method is included directly in the calling method. This avoids the overhead of performing a method call.
- ▶ When reading and writing small amounts of data, use of the Java Buffered I/O classes can significantly improve performance, by minimizing the number of actual I/O calls that need to be made.
- ▶ Avoid the overuse of exceptions. Throwing and catching exceptions can be expensive. Exceptions should be mainly used for (infrequent) error conditions, not as a normal mechanism for control flow by the application. Although throwing and catching exceptions should be minimized, checking for them with the use of `try {} catch {}` blocks is not particularly expensive so this can be used extensively if required. However, avoid catching an exception so that it can be simply rethrown and caught again later. Catch the exception at the level where it will be handled. Also beware of printing stack traces for exceptions, because they can be quite large.
- ▶ It is best to avoid spawning of new threads. Spawned threads do not have access to J2EE resources such as JNDI, security or transaction contexts. Rather than spawning a thread to act as a server to receive incoming messages, consider using message-driven beans (MDBs).
- ▶ In many applications, performance can be improved by performing some caching of data by the application. Note that if this is done, consideration must

be given to periodically flushing the cache to avoid it growing continuously. Also be careful with making assumptions about requests for a client always being served by a particular application server instance. Even if session affinity is used, in a failover situation, HTTP requests can be serviced by a different application server instance, which may not have the cached data. We recommend that the cache be implemented using a well-defined interface, and that data that is not in the cache be retrieved again, transparent to the rest of the application.

**Note:** You can use WebSphere Dynamic Cache service to intercept calls to cacheable objects and store their output in a Dynamic Cache. Refer to Chapter 14, “Dynamic caching” on page 527 for further information on this topic or to the content related to dynamic caching in the WebSphere InfoCenter at

[http://publib.boulder.ibm.com/infocenter/wasinfo/topic/com.ibm.websphere.nd.doc/info/ae/ae/tprf\\_dynamiccache.html](http://publib.boulder.ibm.com/infocenter/wasinfo/topic/com.ibm.websphere.nd.doc/info/ae/ae/tprf_dynamiccache.html)

## 18.10 Reference

For more information, see the following:

- ▶ IBM developerWorks at:

<http://www.ibm.com/developerworks/>  
<http://www.software.ibm.com/wsdd/>

- ▶ WebSphere Best Practices are at:

<http://www.ibm.com/developerworks/websphere/zones/bp/index.html>  
<http://www.ibm.com/developerworks/websphere/zones/bp/background.html>

- ▶ White paper “WebSphere Application Server Development Best Practices for Performance and Scalability”, by Harvey W Gunther, 7 September 2000 found at:

[http://www.ibm.com/software/webservers/appserv/ws\\_bestpractices.pdf](http://www.ibm.com/software/webservers/appserv/ws_bestpractices.pdf)

- ▶ Java Performance Tuning Web site at:

<http://www.javaperformancetuning.com/>

- ▶ *Performance Analysis for Java Web Sites*, by Joines, Wilenberg, Hygh, published by Addison-Wesley, 2002, ISBN 0-201-84454-0.





# Performance tuning

This chapter discusses tuning an existing environment for performance. While not providing specific tuning values, which would be unique to each environment and application, instead this section provides a guide for testing application servers and components of the total serving environment that should be examined and potentially tuned to increase the performance of the application.

This chapter consists of four main parts:

- ▶ “Testing the performance of an application” on page 822
- ▶ “Tools of the trade” on page 823
- ▶ “Performance monitoring guidelines” on page 835
- ▶ “Performance tuning guidelines” on page 843

## 19.1 Testing the performance of an application

Application performance testing is an important component of the software deployment cycle. It becomes even more important with respect to Web applications, as an end user's tolerance for slow applications is generally much lower than that of a captive internal audience. Performance testing has been a subject of many products and white papers recently, and has spawned a new IT industry dedicated to Web application testing.

### 19.1.1 Introduction to application performance testing

When performance testing a Web application, several requirements must be determined either through interpretation of data from an existing application that performs similar work, or from best-guess estimates. Those requirements are:

- ▶ User base

What is the expected number of users that will access this application? This is generally expressed in hits per month, day, hour, or minute depending on volumes.

- ▶ Total concurrent users

During a peak interval, what is the maximum possible number of users accessing the application at the same time? This should be planned for, expected, and re-evaluated on a regular basis.

- ▶ Peak request rate

How many pages will need to be served per second? This also should be re-evaluated regularly.

The user base, especially for Web applications, is a difficult number to determine, especially if this is an application that is performing a new function on a Web site. Use existing Web server measurements to provide a “best-guess” number based on current traffic patterns for the site. If capturing these numbers is not possible, then the best option is to determine the breaking point for the application within the intended deployment infrastructure. This provides the ability to monitor once the application is live, and to provide increased capacity prior to a negative user response due to load.

When dealing specifically with applications running in WebSphere Application Server, there are performance testing protocols that can be followed. A good example of this can be found in the article “Performance Testing Protocol for WebSphere Application Server-based Applications” by Alexandre Polozoff found at:

[http://www7b.software.ibm.com/wsdd/techjournal/0211\\_polozoff/polozoff.html](http://www7b.software.ibm.com/wsdd/techjournal/0211_polozoff/polozoff.html)

One important point to remember is that performance tuning is more of an “art” than a science. Now do not despair if you get the impression that you will not be able to achieve your tuning goals if you believe to lack the “talent” for that art! Because, on the other hand, performance tuning should also be seen as an art that strictly follows the recurring, monotonous trifold process of Testing - Evaluating - Tuning. This process requires that you know your system, your environment and application very well, you know what you want to test, what goal(s) you want to achieve, and also that you know the tools you are going to use for load testing - all of which implies more of a solid handicraft than an artist’s creative work. But having a bit of intuition and developing a feeling for your work will most certainly help you in finding the best configuration more easily. Finally said, experience will be your most valuable friend, so get to know different load testing tools on various environments to gain comprehensive, in-depth knowledge.

The next thing to strictly keep in mind is that it is impossible to tune the environment to significantly impact poor coding practices. It is also important to remember that performance tuning is an ongoing process. Continual reviews of performance should be done to assure that changes in load, application behavior, and site behavior have not adversely impacted the application performance. Also, there are no hard rules for performance tuning. What may be appropriate tuning for one application may not be appropriate for another. It is more important to understand the concepts associated with tuning, and make adjustments based on the understanding gained from those concepts.

## **19.2 Tools of the trade**

The following is an overview of a small choice of load testing tools. It is by no means complete, nor does it imply a recommendation or endorsement of these tools. In the author’s opinion the most comprehensive performance testing tools are (in no special order) Mercury LoadRunner, Rational TestStudio® and Segue SilkPerformer. These tools have a very broad and deep functionality but they are also quite expensive, while for your special case some other (possibly open-source licensable) tool might do the job as well. “ApacheBench” on page 824 and “OpenSTA” on page 827 gives a brief introduction to these two open-source load testing packages, while in “Other testing tools” on page 835 you will find a few links to additional performance testing products and projects.

### **19.2.1 WebSphere Performance Tools**

Web Performance Tools (formerly known as AKtools) were developed as an internal project for IBM to provide a low-cost, highly configurable load-testing tool for Web sites. At this time they are no longer available from the alphaWorks Web

site. The Web Performance Tools have been retired because they overlap with the functionality provided by other IBM tools. These tools are:

- ▶ Rational Suite® TestStudio

<http://www.ibm.com/software/awdtools/suite/>

- ▶ IBM WebSphere Studio Workload Simulator

<http://www.ibm.com/software/awdtools/studioworkloadsimulator/>

## 19.2.2 ApacheBench

ApacheBench (AB) is a very simple tool for quickly generating HTTP GET and POST requests. It is part of the Apache HTTP Server and the IBM HTTP Server powered by Apache. It is automatically installed with IBM HTTP Server and the executable can be found in <IHS\_HOME>\bin\ab. For a manual and list of options, refer to:

<http://httpd.apache.org/docs/programs/ab.html>

**Note:** We have noticed that the AB executable is not part of every IBM HTTP Server distribution: it is not included in the currently available version 1.3.28, but it is again delivered with version 2.0.42.2.

However, you can always download the Apache HTTP Server source-code from the following Web site and compile the AB utility yourself:

<http://httpd.apache.org/download.cgi>

ApacheBench was mainly designed as a benchmarking tool, and it is only usable for very basic load testing because of several limitations:

- ▶ It has no functionality to record a browser click stream and requests have to be specified on the command line.
- ▶ Although the HTTP POST method is supported, POST request data has to be put into a file before the request is sent.
- ▶ There is no support for distributed tests, nor does it provide any graphical representation of the test result data.
- ▶ It cannot be used for scripting of scenarios or testcases, and can merely test one URL at a time.
- ▶ It cannot retrieve cookie data from the response, although cookie information can be specified on the command line, making session-aware performance testing possible in a very limited way.



**Attention:** Be especially aware of one AB limitation: it does not verify the HTTP response!

For example, if your application server returns a “500 Internal Server Error” in 5 ms for a page request you might get the impression that your system is super fast! Always do a test run first with the `-v` (verbose mode) option enabled to have the response code printed out!

Nevertheless, AB has several advantages: It is free, very easy and quick to use, and there is no additional setup required as it is part of IBM HTTP Server. It can be useful for very simple and ad-hoc performance tests, where the goal is for example to stress a Web server or application server to get a rough idea of how many requests (of one kind) can be processed per second. Another use can be to stress the application server, creating a multitude HTTP sessions at a time, while using Tivoli Performance Viewer to monitor the servers’ performance. The example on page 826 shows a sample ApacheBench run stress testing the Trade3 Web primitive PingJSP.

**Important:** Be careful when selecting the URL you wish to test with AB. As you can only specify one URL at a time as the input parameter, make sure to select the URL you really mean to test!

Sometimes a Webpage (in our example the Trade3 main URL `http://<your_host>/trade/`) will provide a detailed display view which is generated in several steps:

1. First the browser retrieves the response from the server which merely contains a frameset or a redirect, and will then:
2. Create additional requests to receive the actual content, like images, and so forth.

ApacheBench is by no means a browser and does not know about framesets, links, images, or dynamic content like Javascript; it will not follow any redirects or even resolve simple HTML `<img src>` tags to retrieve images. This behavior will certainly falsify your performance results if not minded! Following our Trade3 example, be sure to use a request URL that returns meaningful data — not only redirects or framesets — from the application server.

For example, use:

- ▶ `http://<your_host>/trade/PingJsp.jsp`
- ▶ `http://<your_host>/trade/scenario`

## Command line options

The most common and useful options for AB are the following:

<b>-h</b>	Displays help and usage.
<b>-n requests</b>	The (absolute) number of requests to perform for the benchmarking session. The default is 1. Keep in mind that AB does not implement <i>thinktime</i> ! If you set the number of requests too high then you will effectively perform a denial-of-service attack on your server.
<b>-c concurrency</b>	The number of simultaneous requests to perform (virtual users). The default is 1, which means no concurrency.
<b>-v level</b>	Used to set the verbosity level. Level 2 and higher will print HTTP headers and the HTTP response body.
<b>-A username:pass</b>	Used to supply basic authentication credentials to the server.
<b>-C name=value</b>	Add a cookie-header with name and value to the requested object.

**Note:** The number of requests is not set per concurrent user. If you specify -n 20 and -c 10, each of the 10 virtual users will only perform two requests!

As an example, the command for stress testing the Trade3 primitive PingJSP using ApacheBench is:

```
C:\Program Files\IBM HTTP Server 2.0\bin> ab -g test -c 5 -n 50  
"http://cluster.itso.ibm.com/trade/PingJsp.jsp"
```

In this example we perform 50 requests using five virtual users, that is each of the virtual users performs 10 requests.

The results of an ApacheBench stress test returns values such as:

- ▶ Time taken for tests
- ▶ Requests per second
- ▶ Time per request (mean)
- ▶ Time per request (mean, across all concurrent requests)
- ▶ Transfer rate (Kbytes/sec)
- ▶ Percentage of the requests served within a certain time
- ▶ and more

### 19.2.3 OpenSTA

The *Open System Testing Architecture* (OpenSTA) is a distributed software testing architecture based on CORBA. It is open source software licensed under the GNU General Public License. OpenSTA is designed to generate realistic heavy loads simulating the activity of hundreds to thousands of virtual users. This capability is fully realized through OpenSTA's distributed testing architecture. OpenSTA graphs both virtual user response times and resource utilization information from all server systems under test, so that precise measurements can be gathered during load tests and analysis on these measurements can be performed.

#### Feature overview

The name OpenSTA stands for a collection of tools that build on the distributed architecture. These tools allow you to perform scripted HTTP load or stress tests including performance measurements run from a single machine or distributed and coordinated across many machines.

The software package can be obtained from <http://opensta.org/>. There is also an excellent community site for additional information and documentation to be found on <http://portal.opensta.org/>.

The sourcecode can be obtained from:

<http://opensta.sourceforge.net/>

The current version is OpenSTA 1.4.2 and contains the following features:

- ▶ Intelligent script recording: automatic cookie-based session support in recorded scripts
- ▶ Script modeling in a BASIC like Script Control Language (SCL)
- ▶ Script debugging using single-step mode and breakpoints
- ▶ Modular test-suite creation
- ▶ Supported protocols: HTTP 1.0, HTTP 1.1, HTTPS
- ▶ Single machine based load generation
- ▶ Distributed load generating across multiple machines with one controller
- ▶ Support for controlling load generation over the Internet
- ▶ Extensive data collection and simple statistics graphs provided
- ▶ Collected data can be exported into comma separated text files
- ▶ Additional performance measurement data collected through Simple Network Management Protocol (SNMP) and Windows NT performance monitor
- ▶ Tutorial, user's guide and detailed online help available

- ▶ Online community and Web-based discussion forum, see:  
<http://portal.opensta.org/>
- ▶ Commercial support and services available
- ▶ Supported load generating platforms: Windows NT and Windows 2000
- ▶ Easy to use, but more powerful features require manual script modelling (for example, form-based login)

## Prerequisites and installation

We were using the current OpenSTA Version 1.4.2 to test our sample topology described in Chapter 7, “Implementing the sample topology” on page 255. The listed requirements are for that version only. See the latest release notes and the download package for the most up-to-date system requirements, found at:

<http://www.opensta.org/download.html>

### Software requirements

- ▶ Windows 2000
- ▶ Windows NT with service pack 5
- ▶ Microsoft Data Access Components (MDAC) version 2.5 (minimum, can be obtained from <http://microsoft.com/data/download.htm>)

### Supported Web browsers for script recording

- ▶ Microsoft Internet Explorer 4, 5, and 6
- ▶ Netscape Navigator 4.7

**Note:** Other browsers (for example, Opera, etc.) can be used for script recording, but their proxy settings have to be configured manually instead of being configured automatically by the OpenSTA Script Modeler.

### Installation

Install OpenSTA by running the setup wizard and following on-screen directions.

## The OpenSTA architecture

OpenSTA supplies a distributed software testing architecture based on CORBA which enables you to create and run tests across a network. For a detailed overview of the architecture and its components, refer to the *Getting Started Guide* and the *Users Guide* in the documentation section of <http://opensta.org>.

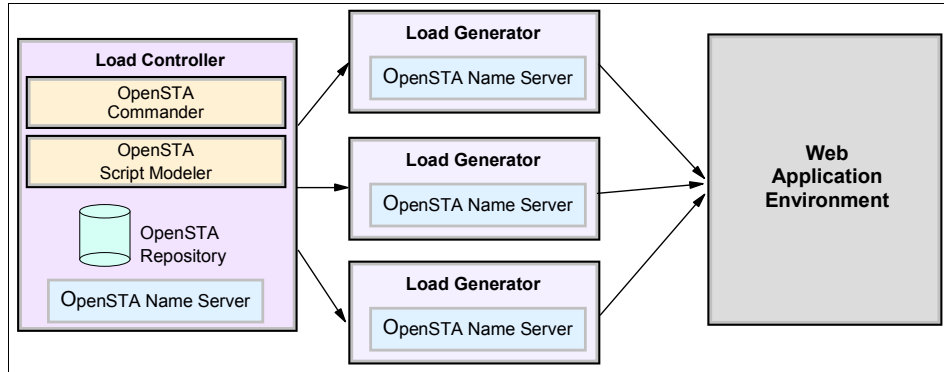


Figure 19-1 Overview of the OpenSTA distributed testing architecture

### OpenSTA Name Server

The *OpenSTA Name Server* configuration utility is the component that allows you to control your test environment. After installation you will notice the OpenSTA Name Server running which is indicated by an icon in the Windows task bar. For a distributed test every load generating node has to have the OpenSTA Name Server installed and configured to point to the controlling node, also called the *repository host*.

### OpenSTA Commander

The *Commander* is the graphical user interface that functions as the front end for all test development and test coordination activity. This is the main component of OpenSTA you need to run in order to perform load tests. In the left pane it shows a view of the OpenSTA *repository*, containing all defined tests, scripts and collectors.

### OpenSTA Script Modeler

The *Script Modeler* is the recording and script developing environment of OpenSTA. It is launched through the Commander by double-clicking a script name in the repository tree window.

## Recording the Trade3 script

With the Script Modeler you can record and/or develop and refine your scripts. They form the content of your tests and let you generate the desired Web activity during a load test. They are stored in the repository from where they can be included into multiple tests. Perform these steps to record a new script:

1. Create a new, empty script by selecting **File -> New Script -> HTTP**, or by right-clicking the **scripts** node in the repository tree window and selecting **New Script -> HTTP** from the pop-up menu. Enter the name **Trade3Scenario**.

2. Launch the Script Modeler by double-clicking the previously created script.
3. Select **Options -> Browsers** and select the browser you want to capture with.
4. Select **Capture -> Record** or alternatively click the **record icon** (the red circle) in the toolbar to start recording. The settings that allow automatic cookie handling are enabled by default. For more information on recording options refer to the online help.

**Note:** OpenSTA provides a proxy gateway that intercepts all HTTP traffic during recording. Script Modeler will automatically configure your browser's proxy setting to point to that gateway, and reset it to the previous values after recording. By default the proxy gateway uses port 81 on the local host.

5. After the browser window has launched, open the URL:

```
http://<host_name>/trade/scenario
```

In our case:

```
http://www.itso.ibm.com/trade/scenario
```

Do not reload the page, as we only want to capture exactly one request. When the page has loaded completely, either close the browser window, or switch back to the Script Modeler and select **Capture -> Stop** from the menu; both will terminate your recording session.

**Tip:** If you need to navigate to a specific page on your site first, switch back to the Script Modeler window and press the **Pause** button in the toolbar. This will temporarily stop recording. When you have navigated to the link you want to start your script with, press the **Pause** button again to resume recording.

6. Save your script and exit Script Modeler.

You have now created a simple user session in the Trade3 application. An extract from that script is shown in Example 19-1. You can now view, replay and modify the recorded script inside Script Modeler. In case you change the script manually use the compile icon on the toolbar to find syntax errors.

*Example 19-1 An excerpt of the recorded OpenSTA script "Trade3Scenario"*

---

```
Start Timer T_TRADE3SCENARIO
```

```
PRIMARY GET URI "http://www.itso.ibm.com/trade/scenario HTTP/1.1" ON 1&
  HEADER DEFAULT_HEADERS&
  ,WITH {"Accept: */*",&
    "Accept-Language: en-us", &
```

```

        "Connection: Keep-Alive"}

Load Response_Info Header on 1&
    Into cookie_1_0&
    ,WITH "Set-Cookie,JSESSIONID"

WAIT 6800

!Cookie : JSESSIONID=00011bubQp5N91qYmlb8KyynR58:v4fh5hfu
GET URI "http://www.itso.ibm.com/trade/style.css HTTP/1.1" ON 2    &
    HEADER DEFAULT_HEADERS&
    ,WITH {"Accept: */*",&
        "Referer: http://www.itso.ibm.com/trade/scenario", &
        "Accept-Language: en-us",&
        "Connection: Keep-Alive",&
        "Cookie: "+cookie_1_0}

DISCONNECT FROM 1
DISCONNECT FROM 2
SYNCHRONIZE REQUESTS
End Timer T_TRADE3SCENARIO

```

---

## Randomizing request using variables

A randomization of the requests is quite easily achieved using variables. All you need is the script editor and a little knowledge of the OpenSTA scripting language. You could for example insert two variables (user and password information) that get filled with different account data on each request, and build a new HTTP GET string using those variables. The variable information can be read by the scripting engine from a flat file, thus effectively simulating different, concurrently active user sessions.

## Additional Script Modeler features

Following is a selection of additional, very useful script modelling features:

- ▶ **Custom timers:** To evaluate the response time for business transactions, manual timers can be inserted into the scripting code.
- ▶ **Output stream parsing:** Verification of the response header and body can be done through manually parsing the output stream using the OpenSTA scripting language.

## Performance testing using the OpenSTA Commander

Using the Commander you can define tests; each test can consist of one or more *task groups*, which are sequences of test scripts. Each task group has to contain at least one script. Once a script has been associated with a task group, it is called a *task*.

Test execution settings like the number of iterations or the number of virtual users (VU) can be specified per task group, while the number of iterations per task is also configurable. This modular structure makes the scenario easy to accomplish, where a user logs in first, then uses the Trade3 application a configurable number of times, and finally logs out (see Example 19-2).

*Example 19-2 A exemplary modular test design in OpenSTA*

---

Test Trade3 will be repeated 10 times, consisting of:

```
|
|--Task group Trade3_1: 50 VU
|   |--Task-1: T3_Login, repeat count: 1
|   |--Task-2: T3_Scenario-1, repeat count: 500
|   |--Task-3: T3_Scenario-2, repeat count: 250
|   |--Task-4: T3_Logout, repeat count: 1
```

---

To run a performance test using OpenSTA perform these steps:

1. Create the test
2. Specify runtime parameters
3. Execute the test
4. View the results

### **Create a test**

The first step is to create a new test and add the previously recorded script:

1. In the Commander window, select **File -> New Test -> Tests** and enter **Trade3** as the name of the new test.
2. Drag the script **Trade3Scenario** from the repository view on the left side onto the **Task1** column on the right side of the Commander window. This creates a new task group called Trade3\_1.

### **Specify runtime parameters**

The next step is to configure the number of iterations for the task and the number of virtual users that will concurrently perform the scripted task. In this example, we will use 40 virtual users performing 10 iterations of the Trade3Scenario task.

1. Click **Trade3Scenario** in column Task1. Change the value in the Number of times each user will run this task (iterations) field to **10**.
2. Click the field containing 1 in column VUs. Enter **40** for the Total number of virtual users for this task group.

### **Additional OpenSTA Commander features**

*Collectors*, which gather operating system performance data like CPU utilization, network and disk I/O, etc., can be defined in the OpenSTA Commander. These collectors will gather data during the test execution and can then be correlated



with the actual test results. The two supported collector schemes to gather data from are:

- ▶ Performance data available from the Windows System Monitor (Windows Management Instrumentation WMI)
- ▶ Simple Network Management Protocol (SNMP) data from SNMP enabled hosts or devices

### ***Execute a test***

The final step is to start the test using the OpenSTA Commander; while it is executing, the test's progress and statistics can be viewed. First you have to change the monitoring interval to five seconds. That way the statistics will be updated every five seconds (in a real world scenario the monitoring interval should be set even higher as to not influence the performance test by wasting CPU and network resources used for collecting performance data).

1. Set the monitoring interval to five seconds:
  - a. Select the **Monitoring** tab.
  - b. Click the **Tools** symbol.
  - c. Set both Task Monitoring Interval and Sampling Frequency to **5**.
2. To start the test select **Test -> Execute Test** or press the green **Play** icon on the toolbar. During the test you can watch the progress and test statistics:
  - a. Switch to the monitoring view by selecting the **Monitoring** register.
  - b. Enable the Summary checkbox on the right side of the Commander window to see the following current statistics:
    - Active virtual users
    - Test duration
    - HTTP requests/second
    - Number of successful/failed HTTP requests

Right-click inside the Summary window to select and deselect additional statistics data.

The test will stop automatically after all 40 VUs have cycled through their 10 iterations of the Trade3Scenario task.

### **View results**

After the test has stopped and results data has been collected, switch to the Results view tab to examine the test data and statistics. There you can view different reports and even some graphs.

The most useful reports and graphs are:

- ▶ Test Summary, Audit Log, Report Log, and Test Error Log

- ▶ HTTP Data List (for debugging HTTP responses)
- ▶ HTTP Response Time (Average per Second) versus Number of Responses Graph  
This graph displays the average response time for requests grouped by the number of requests per second during a test run.
- ▶ HTTP Errors versus Active Users Graph  
This graph displays the effect on performance measured by the number of HTTP server errors returned as the number of active Virtual Users varies during a test run.
- ▶ HTTP Responses versus Elapsed Time Graph  
This graph displays the total number of HTTP responses per second during the test execution.
- ▶ HTTP Response Time versus Elapsed Time Graph  
This graph displays the average response time per second of all the requests issued during the test run.
- ▶ Timer Values versus Active Users Graph
- ▶ Timer Values versus Elapsed Time Graph

OpenSTA Commander also provides two additional features to help you with further data analysis:

- ▶ Exporting results data  
This feature allows you to export every text-based report into a comma-separated-values (CSV) file. Every graphics report can be exported directly into Microsoft Excel, where you can perform additional statistical calculations. Export the data by right-clicking inside the opened report window, select **Export** and save the CSV-file or open it inside Excel.
- ▶ URL filtering the report data  
This feature is available when right-clicking inside a graphics report and selecting **Filter URLs**. For example, you can filter out any URLs so that only the servlet request to /trade/scenario is left. Then the graph displays only the response times for the actual dynamic content of your load test.

The most interesting reports are:

- ▶ HTTP Data List
- ▶ HTTP Response Time versus Elapsed Time
- ▶ HTTP Responses versus Elapsed Time

## 19.2.4 Other testing tools

There are a myriad of open-source or commercial products and services related to Web application testing. A search on Google for Web Application Testing can yield in excess of two million hits. Below is a list of just some of the commercial and free tools available. This list is by no means complete, and does not imply an endorsement or recommendation of the tool. It is merely provided as an alternative source of testing tools if ApacheBench or OpenSTA are not used.

- ▶ **JMeter**, Open Source software available from the Apache Software Foundation at:  
<http://jakarta.apache.org/jmeter/>
- ▶ **TestMaker** and **TestNetwork**, from PushToTest. See:  
<http://www.pushtotest.com/ptt/>
- ▶ **Grinder**. See:  
<http://grinder.sourceforge.net>
- ▶ **LoadRunner** from Mercury Interactive Corporation. See:  
<http://www.merc-int.com>
- ▶ **IBM Rational Suite TestStudio** from IBM. See:  
<http://www.ibm.com/software/awdtools/suite/>
- ▶ **Segue SilkPerformer**. See:  
<http://www.segue.com/products/load-stress-performance-testing/index.asp>
- ▶ **WebLOAD** from Radview. See:  
<http://www.radview.com>
- ▶ **WebStone** from Mindcraft. See:  
<http://www.mindcraft.com>
- ▶ **OpenLoad** from Opendemand Systems. See:  
<http://www.opendemand.com/openload/>

## 19.3 Performance monitoring guidelines

Before tuning measures can be taken, there is one important question that has to be answered first: “What part of the system shall I tune?”. Simple random tuning acts will seldom give the desired effect, and sometimes will even produce worse results. This section discusses which components to examine first, and gives a practical hotlist of top-ten monitors and which tools to use. Finally it makes a short excursion into performance analysis and load testing best practices.

## 19.3.1 Monitoring using Tivoli Performance Viewer and Advisors

If you experience performance problems, start using Tivoli Performance Viewer (TPV) and walk through the following top-ten monitoring items checklist. It will help you to more easily find those significant parts of your system where the most performance impact can be gained by tuning. It cannot be stated often enough that performance tuning is not a science, but an art, so do not expect miracles by adjusting a few “knobs” inside WebSphere Application Server: every system and every application behaves differently, and requires different tuning measures! But the Performance Monitoring Infrastructure (PMI), Tivoli Performance Viewer and the Performance Advisors will assist you on the way to achieve your goal, a system configured for optimum performance.

### Top-ten monitoring hotlist

Because of the sheer magnitude of monitors and tuning parameters, knowing where to start, what to monitor and which component to tune first is still hard. Follow these top-ten monitoring steps to check the most important counters and metrics of WebSphere Application Server. See Figure 19-2 on page 837 for a graphical overview of the most important resources to check. Consider the following:

- ▶ If you recognize something out of the ordinary, for example, an overutilized thread pool or a JVM that spends 50% in garbage collection at peak-load time, then concentrate your tuning actions there first.
- ▶ Perform your examination when the system is under typical, production level load and make notes of the observed values.
- ▶ Alternatively, save Tivoli Performance Viewer sessions to the filesystem, where the monitoring data will be stored for recurring analysis.
- ▶ Keep in mind that one set of tuning parameters for one application will not work the same way for another.

For tips and good starting points on which direction to tune a specific parameter, refer to 19.4, “Performance tuning guidelines” on page 843 for respective tuning measures related to this checklist.

### ***Servlets and Enterprise Java Beans***

1. Average response Time
2. Number of requests per second (Transactions)
3. Live number of HTTP Sessions

### ***Thread pools***

4. Web server threads
5. Web container and EJB container thread pool
6. Datasource connection pool size

## Java Virtual Machine

7. Java Virtual Machine (JVM) memory, garbage collection statistics

## System resources on Web, application, and Database servers

8. CPU utilization

9. Disk and network I/O

10. Paging activity

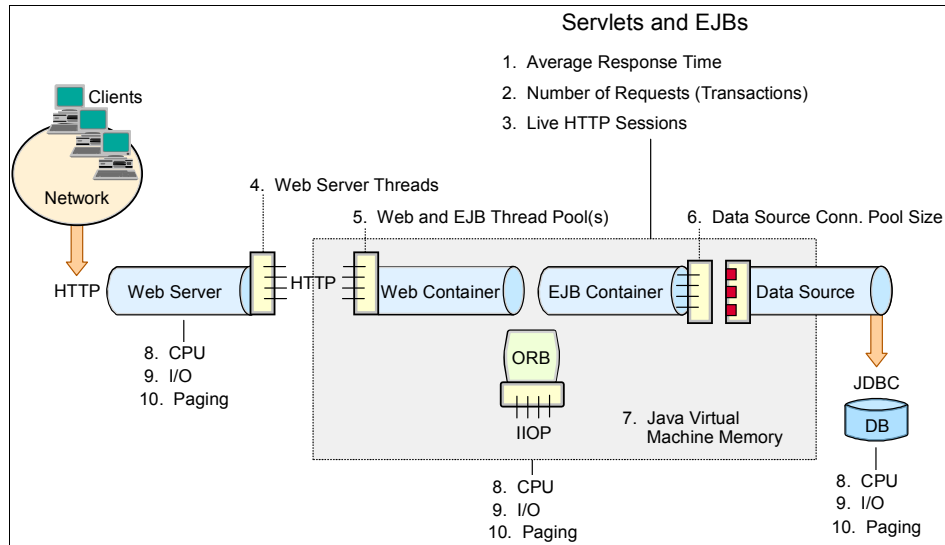


Figure 19-2 Top-ten monitoring items checklist

## Tivoli Performance Advisor

Use Tivoli Performance Advisor (TPA) in addition to TPV, together these tools will help you to tune your system, and Tivoli Performance Advisor will give you recommendations on inefficient settings.

View recommendations and data from the Performance Advisor by expanding the Performance Advisor icon under Data Collection in Tivoli Performance Viewer and selecting the server, respectively. For detailed instructions on how to use the Performance Advisor, refer to 16.11, “Performance Advisors” on page 745.

**Hint:** Sometimes TPA complains that you should set the instrumentation level for JVM to maximum. If you have already done that and you still see that message, what you actually need to do is to activate the JVMPi facility. See 16.2.7, “Using JVMPi facility for PMI statics” on page 701 for instructions how to enable JVMPi logging.

## Runtime Performance Advisor

The Runtime Performance Advisor (RPA) service runs inside each application server process and has to be enabled explicitly. It basically uses the same rules engine as Tivoli Performance Advisor and will produce the same tuning recommendations. Since it has a only a very small performance impact, it can be enabled and left running for a longer period of time even during production if you experience performance problems. This way you can get important system tuning information and advice even when you are not actively monitoring the system by using Tivoli Performance Viewer and Advisor. For a detailed explanation on using the Runtime Performance Advisor, refer also to 16.11, “Performance Advisors” on page 745.

**Tip:** It is important to configure the number of processors in the Runtime Performance Advisor configuration panel, otherwise the recommendations can be inaccurate.

However, keep in mind that both TPA and RPA advice can only be accurate if your application runs with production level load, and without exceptions and errors. This is best done by running a performance test, where a production level load can be simulated in a reproducible manner. Both advisors will need the CPU utilization to be high to provide advice, which is best accomplished during a load and stress test.

### 19.3.2 Performance analysis

Web performance analysis describes the process of finding out how well your system - a Web application or generally a Web site - performs, and pinpointing performance problems caused by inadequate resource utilization, such as memory leaks or over- or underutilized object pools, to name just a few. Once you know the trouble spots of your system, you can take counter-measures to reduce their impact by taking appropriate tuning actions.

#### Terminology

Performance analysis is a quite comprehensive topic. It fills whole books and is surely out of scope of this book. What is attempted here is to give a short introduction into that subject. Following is a concise definition of the three most important concepts used in performance analysis literature:

- ▶ Load
- ▶ Throughput
- ▶ Response Time

### ***Load***

A Web site - and especially the application that is running behind it - typically behaves and performs different depending on the current load, that is, the number of users that are concurrently *using* the Web site, at one point in time. This includes clients who actively perform requests at a time, but also clients who are currently reading a previously requested Web page. Peak load often refers to the maximum number of concurrent users using the site within some point in time.

### ***Throughput***

A Web site can only handle a specific number of requests in parallel. Throughput depends on that number and on the average time a request takes to process; it is measured in requests/second. If the site can handle 100 requests in parallel and the average request takes one second, the Web site's throughput is 100 requests per second.

### ***Response Time***

Response time refers to the timeframe from when the client initiates a request until it receives the response. Typically the time taken to display the response (usually the HTML data inside the browser window) is also accounted for in the response time.

## **Performance testing**

To find out the performance of your system in the first place, you will have to run a performance test to get an idea of how many users will be able to access the site at the same time without noteworthy delays. Even at a later time, load tests are most helpful to find out how much performance was gained by a specific tuning measure. After each tuning step, the load test has to be repeated (10-15 iterations of it are not uncommon), until the system is tuned for optimal performance. For performance assessment and tuning, the following test requirements can be stated:

- ▶ The load expected on the system has to be definable.  
  
That implies that you have to create a model of your expected real-world, production level workload, based on your experience and knowledge of your clients' behavior. Using this representative model in combination with a stress test tool, you will create as many testcases as needed, and combine them into a testsuite that simulates the desired user behavior and performs the necessary interactions with your site during a load test. In some cases, when no good estimate of such a model can be given, even a crude approximation is better than performing no load test at all, and it can always be refined for future test cycles.
- ▶ The load test has to be repeatable.

Tuning without the ability to perform repeatable tests is very hard, because the lack of comparable performance data. To be repeatable, it is necessary to restore the initial system state after each test iteration. Usually persistent application or transaction data is stored in databases or backend systems, so that implies the following requirement:

A second, staging database and or backend system has to be used for testing purposes. Changes to the data on that system will not have consequences in the real world: meaning, the placing of a thousand orders in an online shop during load testing will not activate the order fulfillment process in the backend.

- Find the optimum load level.

The goal is to find the site's saturation point. First, the system has to be driven even over the point where performance begins to degrade. This is commonly referred to as *drawing the throughput curve* (see Figure 19-3 on page 841).

Start a series of tests to plot the throughput curve. Begin testing with wide open server queues (pools) to allow maximum concurrency in your system. Record the throughput (average requests per second) and the response time (average time for requests), increasing the concurrent user load after each test. You will find the system's saturation point in the diagram where the throughput becomes constant (the maximum throughput point is reached), but response time begins to grow proportionally with the user load.

Refer to 19.4.4, "Adjusting WebSphere Application Server system queues" on page 846 for more information on the queuing network and "Determining optimum queue sizes" on page 848 for detailed explanation of the steps on drawing the throughput curve.

**Note:** Here the goal is to drive CPU utilization to nearly 100 percent. If you cannot reach that state with opened up queues, there is most likely a bottleneck in your application, in your network connection, some hard limit on a resource or possibly any other external component you did not modify.



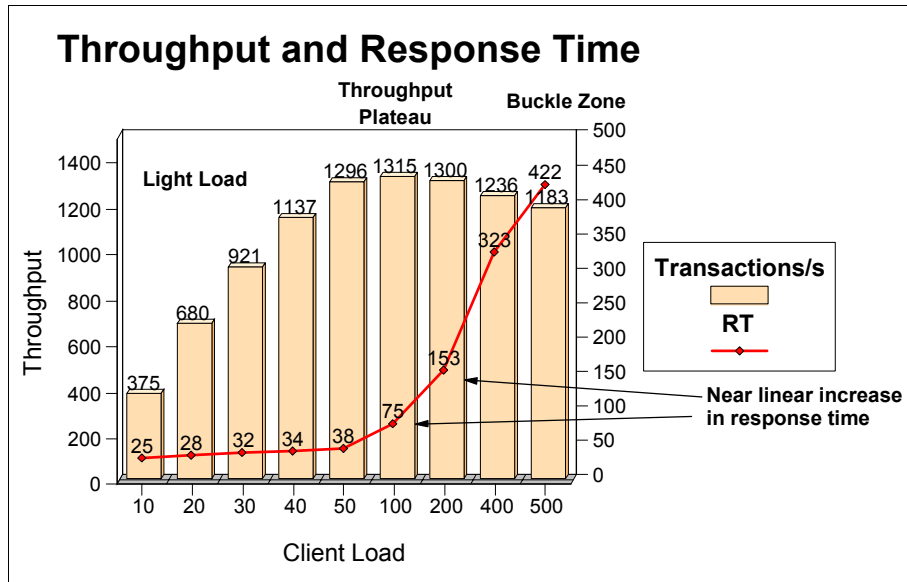


Figure 19-3 How to find the optimum load level for a system: the saturation point

## The Monitoring - Tuning - Testing cycle

Perform an initial load test to get a *baseline* that you can refer to later to, with the Performance Advisor facility enabled. Use the number of users at your saturation point as the load parameter in all future load tests. Check the monitors in TPV according to the top-ten hotlist and the Performance Advisors. After you have implemented an advisor's recommendation, perform another test cycle to find out if you have gained throughput (because of a decreased average response time) or freed resources (for example, memory by reducing thread pools) which you could then possibly spend on other components that need more memory resources. Repeat this procedure to find an optimum configuration for your system and application. Keep in mind that performance tuning is an iterative process, so do not rely on results from a single run.

Here is a short overview of the steps necessary to perform a load test for tuning purposes with Tivoli Performance Advisor as described above:

1. Enable the JVMPI facility for the application server.
2. Enable the PMI service in the application server and Node Agent (if using a WebSphere Network Deployment environment), and restart both.
3. Start Tivoli Performance Viewer and set monitoring levels to Standard.

**Note:** Some JVM rules need instrumentation level set to max. TPA will print a message if it needs a higher monitoring level.

4. Simulate your representative production level load using a stress test tool.
  - Make sure that there are no errors or exceptions during the load test.
  - Record throughput and average response time statistics to plot a curve at the end of all testing iterations.
5. Check Tivoli Performance Advisor and apply advice and follow your own intuition.
  - Restart components or services if necessary.
  - Reset all components (for example, the database) to the initial state.
6. Retest (go back to step 4).

### ***Data capture best practices***

To get accurate results, mind the following best practices for data capturing:

- ▶ Measure during steady-state of the load test

Do not include ramp-up/ramp-down times in your performance data measurements and analysis (see Figure 19-4 on page 843). Measure during the steady-state when the maximum of users are concurrently performing requests.

- ▶ Monitor machine logs and resources

Monitor important log files for exceptions or errors. Be sure that there are no exceptions or deadlocks in the database. Keep an eye on system resources like memory, paging activity, CPU, disk IO, network utilization, socket states, etc., for bottlenecks.

Important log files are SystemOut.log and SystemErr.log. Monitor these logs to make sure your application runs without errors. SystemErr.log should typically remain free of entries. Errors logged there *must* be solved before you can capture meaningful performance data. Likewise, any sort of exception in SystemOut.log during the performance run should be solved before another run, because exception handling and the I/O necessary to write stacks to the log are expensive operations that impact performance.

For a Network Deployment cluster, you don't need to repeat all the monitoring steps for each cluster member if they are all set up identically; monitoring one or two representative cluster members should be sufficient. What is essential however, is to check the CPU statistics for each node to make sure that all cluster member processes are using similar amounts of CPU and there are

no extra processes consuming CPU on any nodes that can interfere with the application server CPU efficiency.

- ▶ Test on quiet systems

Avoid testing during database backups or maintenance cycles.

- ▶ Use isolated networks

Whenever possible, load driving machines should be sharing the same network switch/router as your Web application servers, to rule out additional network latency and network delays.

- ▶ Performance tuning is an iterative process

10-15 test runs are quite usual during tuning phase. Perform long lasting runs to detect resource leaks, for example, memory leaks, where the load tested application runs out of heap space only after a given time.

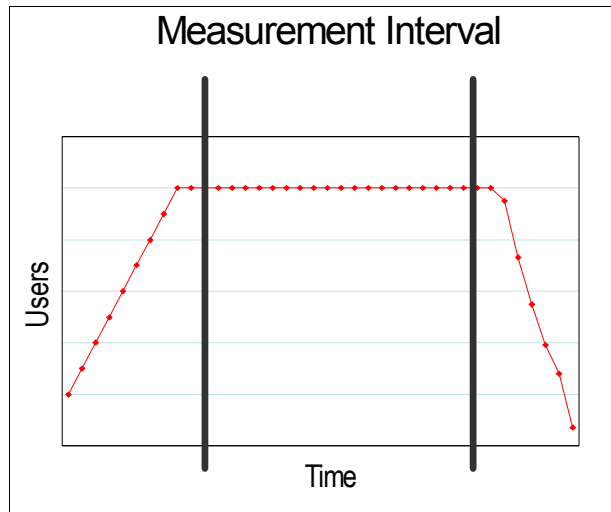


Figure 19-4 Measurement interval: concurrently active users versus elapsed time

## 19.4 Performance tuning guidelines

Tuning is about utilizing resources to their fullest potential, resulting in the fastest request processing possible. Many components in WebSphere Application Server have an impact on performance and tuning is highly application-dependent. This section discusses how to identify bottlenecks, gives a practical introduction into analyzing them, and finally gives recommendations on settings for major WebSphere environment properties.

It is important to stress once again that performance tuning is not an exact science. Factors that influence testing vary from application to application, and also from platform to platform. This section is designed to provide a primer for the reader in a way that describes areas that can be tuned to increase performance.

The latest performance tuning information can be found in the WebSphere Application Server V5.1 InfoCenter, located at

<http://www.ibm.com/software/webservers/appserv/infocenter.html>

Select the appropriate version of the InfoCenter you wish to read. Then navigate to the **Tuning** section in the contents pane on the left side.

### 19.4.1 Tuning parameter hotlist

The following list of tuning suggestions is a subset of the complete list that follows in this chapter. These parameters are on a hot list because they have an important impact on performance. The majority of the parameters are application dependent, and should only be adjusted after close inspection of testing results for the application in question.

- ▶ Hardware and capacity settings, see 19.4.3, “Hardware and capacity settings” on page 845.
- ▶ Java Virtual Machine heap size, see 19.4.6, “Java tuning” on page 870.
- ▶ Application assembly performance checklist, see 19.4.5, “Application assembly performance checklist” on page 863.
- ▶ Data sources connection pool and prepared statement cache, see “Data sources” on page 852.
- ▶ Solaris operating system TCP\_TIME\_WAIT\_INTERVAL, see “Solaris” on page 885.
- ▶ Pass by value versus Pass by reference, see “Pass by value versus Pass by reference (com.ibm.CORBA.iop.noLocalCopies)” on page 903.
- ▶ IBM HTTP Server access logs, see “Access logs” on page 890.
- ▶ HTTP keep-alive connections, see “MaxKeepAliveConnections” on page 858.
- ▶ HTTP transport custom properties, see “HTTP transport custom properties” on page 860
- ▶ Transaction logs, see 19.4.14, “Transaction service settings - transaction log” on page 905.

## 19.4.2 Parameters to avoid failures

The following list of parameters is a subset of the entire list of parameters, and is designed to minimize application failures. The majority of these will also be application specific and should be adjusted based on observed application execution and configuration:

- ▶ Number of Connections to DB2:  
Default settings are likely too low. See “Data sources” on page 852.
- ▶ Allow thread allocation beyond maximum:  
This has been selected and the system is overloaded because too many threads are allocated. See “Thread pool” on page 857 for more information.
- ▶ Using TCP Sockets for DB2 on Linux:  
For local databases. See “Use TCP sockets for DB2 on Linux” on page 907.
- ▶ Connection Pool Size:  
Ensure enough connections for transaction processing with Entity EJBs and for avoiding deadlock. See “Connection pool size” on page 852.

## 19.4.3 Hardware and capacity settings

Obviously, the hardware and its capacity play an important role in performance. Part 1 to Part 4 of this redbook provide detailed information about how to set up a scalable WebSphere environment. However, the following parameters are to be considered:

- ▶ Disk speed  
Disk speed and configuration can have a dramatic effect on the performance of application servers that run applications that are heavily dependent on database support, that use extensive messaging, or are processing workflow. Using disk I/O subsystems that are optimized for performance, for example RAID array, are essential components for optimum application server performance in these environments. It is recommended that you spread the disk processing across as many disks as possible to avoid contention issues that typically occur with one or two disk systems.
- ▶ Processor speed  
Increasing the processor speed often helps throughput and response times once other bottlenecks have been removed where the processor was waiting for such events as input and output, and application concurrency.
- ▶ System memory  
Increasing memory to prevent the system from paging memory to disk improves the performance. Allow a minimum of 256 MB of memory for each

processor (512 MB is recommended). Adjust the parameter when the system is paging and processor utilization is low because of the paging.

► Networks

Run network cards and network switches at full duplex. Running at half duplex decreases performance. Verify that the network speed can accommodate the required throughput. Also, make sure that 100 MB is in use on 10/100 Ethernet networks.

## 19.4.4 Adjusting WebSphere Application Server system queues

WebSphere Application Server establishes a queuing network, which is a group of interconnected queues that represent various components. There are queues established for the network, Web server, Web container, EJB container, Object Request Broker (ORB), data source, and possibly a connection manager to a custom back-end system. Each of these resources represents a queue of requests waiting to use that resource. Queues are load-dependent resources. As such, the average response time of a request depends on the number of concurrent clients.

As an example, think of an application, consisting of servlets and EJBs, that accesses a back-end database. Each of these application elements reside in the appropriate WebSphere component (for example servlets in the Web container) and each component can handle a certain number of requests in a given time frame.

A client request enters the Web server and travels through WebSphere components in order to provide a response to the client. Figure 19-5 illustrates the processing path this application takes through the WebSphere components as interconnected pipes that form a large tube.

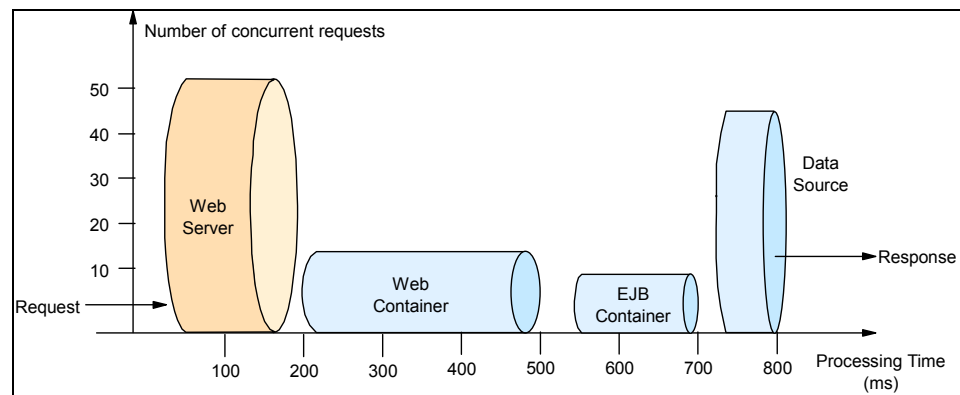


Figure 19-5 Queuing network

The width of the pipes (illustrated by height) represents the number of requests that can be processed at any given time. The length represents the processing time that it takes to provide a response to the request.

In order to find processing bottlenecks, it is useful to calculate a transactions per second (tps) ratio for each component. Ratio calculations for a fictional application are shown in Example 19-3:

*Example 19-3 Transactions per second ratio calculations*

---

The Web Server can process 50 requests in 100 ms =  $\frac{50req}{0.1s} = 500tps$

The Web container parts can process 18 requests in 300 ms =  $\frac{18req}{0.3s} = 60tps$

The EJB container parts can process 9 requests in 150 ms =  $\frac{9req}{0.15s} = 60tps$

The datasource can process 40 requests in 50 ms =  $\frac{40req}{0.05s} = 800tps$

---

The example shows that the application elements in the Web container and in the EJB container process requests at the same speed. Nothing would be gained from increasing the processing speed of the servlets and/or Web container because the EJB container would still only handle 60 transactions per second. The requests normally queued at the Web container would simply shift to the queue for the EJB container.

This example illustrates the importance of queues in the system. Looking at the operations ratio of the Web and EJB containers, each is able to process the same number of requests over time. However, the Web container could produce twice the number of requests that the EJB container could process at any given time. In order to keep the EJB container fully utilized, the other half of the requests must be queued.

It should be noted that it is common for applications to have more requests processed in the Web server and Web container than by EJBs and back-end systems. As a result, the queue sizes would be progressively smaller moving deeper into the WebSphere components. This is one of the reasons queue sizes should not solely depend on the operation ratios.

The following section outlines a methodology for configuring the WebSphere Application Server queues. The dynamics of an individual system can be dramatically changed by moving resources, for example moving the database server onto another machine, or providing more powerful resources, for example

a faster set of CPUs with more memory. Thus, adjustments to the tuning parameters are for a specific configuration of the production environment.

## Queuing before WebSphere

The first rule of tuning is to minimize the number of requests in WebSphere Application Server queues. In general, requests should wait in the network (in front of the Web server), rather than waiting in WebSphere Application Server. This configuration allows only those requests that are ready to be processed to enter the queuing network. To accomplish this, specify that the queues furthest upstream (closest to the client) are slightly larger, and that the queues further downstream (furthest from the client) are progressively smaller.

As an example, the queuing network becomes progressively smaller as work flows downstream. When 200 client requests arrive at the Web server, 125 requests remain queued in the network because the Web server is set to handle 75 concurrent clients. As the 75 requests pass from the Web server to the Web container, 25 remain queued in the Web server and the remaining 50 are handled by the Web container. This process progresses through the data source until 25 user requests arrive at the final destination, the database server. Because there is work waiting to enter a component at each point upstream, no component in this system must wait for work to arrive. The bulk of the requests wait in the network, outside of WebSphere Application Server. This type of configuration adds stability, because no component is overloaded. The Edge Server Components can be used to direct waiting users to other servers in a WebSphere Application Server cluster.

**Note:** If resources are more readily available on the application server or database server, it may be appropriate to tune such that every request from the Web server has an available application server thread, and every application server thread has an available database connection. The need for this type of configuration depends on the application and overall site design.

## Determining optimum queue sizes

A simple way to determine the right queue size for any component is to perform a number of load runs against the application server environment at a time when the queues are very large, ensuring maximum concurrency through the system.

For example, one approach would be:

- ▶ Set the queue sizes for the Web server, Web container and data source to an initial value, for example 100.
- ▶ Simulate a large number of typical user interactions entered by concurrent users in an attempt to fully load the WebSphere environment. In this context, “concurrent users” mean simultaneously active users that send a request,



wait for the response, and immediately resend a new request upon response reception - without thinktime.

Use any stress tool to simulate this workload, such as OpenSTA, discussed in “Testing the performance of an application” on page 822 or the tools mentioned in “Other testing tools” on page 835.

- ▶ Measure overall throughput and determine at what point the system capabilities are fully stressed (the saturation point).
- ▶ Repeat the process, each time increasing the user load. After each run, record the throughput (requests per second) and response times (seconds per request) and plot the throughput curve.

The throughput of WebSphere Application Server is a function of the number of concurrent requests present in the total system. At some load point, congestion will start to develop due to a bottleneck and throughput will increase at a much lower rate until reaching a saturation point (maximum throughput value). The throughput curve should help you identify this load point.

It is desirable to reach the saturation point by driving CPU utilization close to 100%, since this gives an indication that a bottleneck is not caused by something in the application. If the saturation point occurs before system utilization reaches 100%, there is likely another bottleneck that is being aggravated by the application. For example, the application might be creating Java objects causing excessive garbage collection bottlenecks in Java.

**Note:** There are two ways to manage application bottlenecks: remove the bottleneck or replicate the bottleneck. The best way to manage a bottleneck is to remove it. You can use a Java-based application profiler, such as WebSphere Studio Application Developer (see Chapter 17, “Development-side performance and analysis tools” on page 759 for more information), Performance Trace Data Visualizer (PTDV), Optimizelt, JProbe or Jinsight to examine overall object utilization.

The most manageable type of bottleneck occurs when the CPUs of the servers become fully utilized. This type of bottleneck can be fixed by adding additional or more powerful CPUs.

An example throughput curve is shown in Figure 19-6 on page 850.

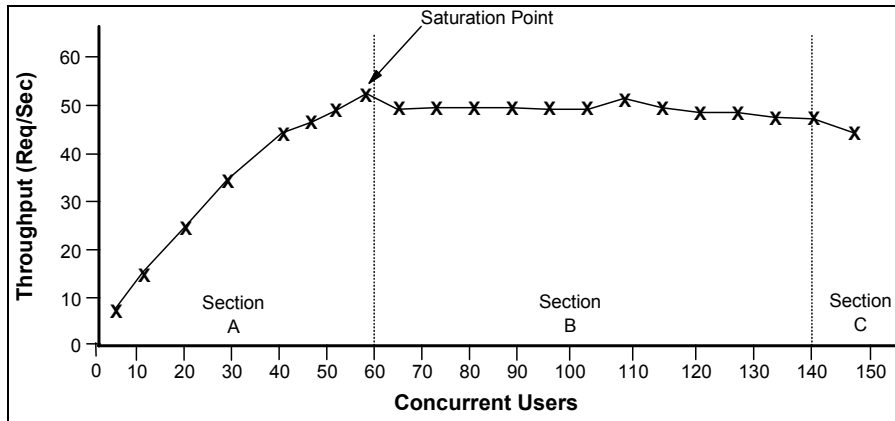


Figure 19-6 Throughput curve

In Figure 19-6, Section A contains a range of users that represent a light user load. The curve in this section illustrates that as the number of concurrent user requests increase, the throughput increases almost linearly with the number of requests. You can interpret this to mean that at light loads, concurrent requests face very little congestion within the WebSphere Application Server system queues.

In the heavy load zone or Section B, as the concurrent client load increases, throughput remains relatively constant. However, the response time increases proportionally to the user load. That is, if the user load is doubled in the heavy load zone, the response time doubles.

In Section C (the buckle zone) one or more of the system components have become exhausted and throughput starts to degrade. For example, the system might enter the buckle zone when the network connections at the Web server exhaust the limits of the network adapter or if the requests exceed operating system limits for file handles.

### Determining the maximum concurrency point

The number of concurrent users at the throughput saturation point represents the maximum concurrency of the application. For example, if the application saturated the application server at 50 users, 48 users might give the best combination of throughput and response time.

This value is called the Max Application Concurrency value. Max Application Concurrency becomes the preferred value for adjusting the WebSphere Application Server system queues. Remember, it is desirable for most users to wait in the network; therefore, queue sizes should decrease when moving downstream farther from the client. For example, given a Max Application

Concurrency value of 48, start with system queues at the following values: Web server 75, Web container 50, data source 45. Perform a set of additional experiments adjusting these values slightly higher and lower to find the best settings.

The Tivoli Performance Viewer (TPV) can be used to determine the number of concurrent users through the Servlet Engine Thread Pool Concurrently Active Threads metric. Refer to 16.3, “Using Tivoli Performance Viewer” on page 705 for more information on TPV.

## **Adjusting queue settings for access patterns**

In many cases, only a fraction of the requests passing through one queue enters the next queue downstream. In a site with many static pages, many requests are fulfilled at the Web server and are not passed to the Web container. In this circumstance, the Web server queue can be significantly larger than the Web container queue. In the previous section, the Web server queue was set to 75 rather than closer to the value of Max Application Concurrency. Similar adjustments need to be made when different components have different execution times. As the percentage of static content decreases, however, a significant gap in the Web server queue and the application server queue can create poorly performing sites overall. Remember that tuning is an art, not a science, and different Web sites have different requirements.

For example, in an application that spends 90% of its time in a complex servlet and only 10% making a short JDBC query, on average 10% of the servlets are using database connections at any time, so the database connection queue can be significantly smaller than the Web container queue. Conversely, if much of a servlet execution time is spent making a complex query to a database, consider increasing the queue values at both the Web container and the data source. Always monitor the CPU and memory utilization for both the WebSphere Application Server and the database servers to ensure the CPU or memory are not being saturated.

## **Configuring the queues**

Within WebSphere Application Server, the queues are represented as pooled resources, for example thread pools or database connection pools. Pool settings determine the maximum concurrency level of a resource. This section describes how the different queues are represented in WebSphere and their settings.

As the queues are most easily tuned from the inside out of the WebSphere Application Server environment, this section describes the queue properties in this order (looking at the components from right to left in Figure 19-5 on page 846).

## Data sources

There are two settings to be concerned with for determining data source queues:

- ▶ Connection pool size
- ▶ Prepared statement cache size

### ***Connection pool size***

When accessing any database, the initial database connection is an expensive operation. WebSphere Application Server supports JDBC 2.0 Standard Extension APIs to provide support for connection pooling and connection reuse. The connection pool is used for direct JDBC calls within the application, as well as for enterprise beans using the database.

Tivoli Performance Viewer can help find the optimal size for the connection pool. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the Pool Size, Percent Used and Concurrent Waiters counters of the data source entry under the JDBC Connection Pools module. The optimal value for the pool size is that which reduces the values for these monitored counters. If Percent Used is consistently low, consider decreasing the number of connections in the pool.

Better performance is generally achieved if the value for the connection pool size is set lower than the value for the Max Connections in the Web container. Lower settings for the connection pool size (10-30 connections) typically perform better than higher (more than 100) settings. On UNIX platforms, a separate DB2 process is created for each connection. These processes quickly affect performance on systems with low memory, causing errors.

Each entity bean transaction requires an additional connection to the database specifically to handle the transaction. Be sure to take this into account when calculating the number of data source connections.

The connection pool size is set from the Administrative Console using these steps:

1. Select **Resources** -> **JDBC Providers** in the console navigation tree.
2. Select the appropriate scope (cell, node or server), depending on your configuration.
3. Open the JDBC provider configuration by clicking the name of the provider.
4. Select the **Data Sources** entry in the Additional Properties pane.
5. Open the data source configuration by clicking the data source name.
6. Select the **Connection Pool** entry in the Additional Properties pane of the workspace.

7. Use the Min connections and Max connections fields to configure the pool size.
8. Save the configuration and restart the affected application servers for the changes to take effect.

The default values are 1 for Min connections and 10 for Max connections.

Deadlock can occur if the application requires more than one concurrent connection per thread, and the database connection pool is not large enough for the number of threads. Suppose each of the application threads requires two concurrent database connections and the number of threads is equal to the maximum connection pool size. Deadlock can occur when both of the following are true:

- Each thread has its first database connection, and all are in use.
- Each thread is waiting for a second database connection, and none would become available, since all threads are blocked.

To prevent the deadlock in this case, the value set for the database connection pool must be at least one higher than the number of waiting threads in order to have at least one thread complete its second database connection.

To avoid deadlock, code the application to use, at most, one connection per thread. If the application is coded to require C concurrent database connections per thread, the connection pool must support at least the following number of connections, where T is the maximum number of threads:

$$T * (C - 1) + 1$$

The connection pool settings are directly related to the number of connections that the database server is configured to support. If the maximum number of connections in the pool is raised, and the corresponding settings in the database are not raised, the application fails and SQL exception errors are displayed in the stderr.log file.

### ***Prepared statement cache size***

The data source optimizes the processing of prepared statements to help make SQL statements process faster. It is important to configure the cache size of the data source to gain optimal statement execution efficiency. A prepared statement is a precompiled SQL statement that is stored in a prepared statement object. This object is used to efficiently execute the given SQL statement multiple times. If the JDBC driver specified in the data source supports precompilation, the creation of the prepared statement will send the statement to the database for precompilation. Some drivers might not support precompilation and the prepared statement might not be sent until the prepared statement is executed.

If the cache is not large enough, useful entries will be discarded to make room for new entries. In general, the more prepared statements your application has, the larger the cache should be. For example, if the application has five SQL statements, set the prepared statement cache size to 5, so that each connection has five statements.

Tivoli Performance Viewer can help tune this setting to minimize cache discards. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the PrepStmt Cache Discard counter of the JDBC Connection Pools module. The optimal value for the statement cache size is the setting used to get either a value of zero or the lowest value for PrepStmt Cache Discards.

As with the connection pool size, the statement cache size setting requires resources at the database server. Specifying too large a cache could have an impact on database server performance. It is highly recommended that you consult your database administrator for determining the best setting for the prepared statement cache size.

**Note:** The statement cache size setting defines the maximum number of prepared statements cached per connection. This is different from WebSphere Application Server V4, where this was specified per container.

The cache size is set from the Administrative Console using these steps:

1. Select **Resources** -> **JDBC Provider** in the console navigation tree.
2. Select the name of the provider from the list of JDBC providers.
3. Select the **Data Sources** entry in the Additional Properties pane.
4. Select the name of the data source.
5. Use the Statement Cache Size field to configure the total cache size.
6. Save the configuration and restart the affected application servers for the change to take effect.

## EJB container

After the EJB container is deployed, you can use the following parameters to make adjustments that improve performance.

### ***Cache settings (Cache size and Cleanup interval)***

To determine the cache absolute limit, multiply the number of enterprise beans active in any given transaction by the total number of concurrent transactions expected. Then, add the number of active session bean instances. Use the Tivoli

Performance Viewer to view bean performance information. The cache settings consist of two parameters: the cache size and the cleanup interval.

The cleanup interval specifies the interval at which the container attempts to remove unused items from the cache in order to reduce the total number of items to the value of the cache size.

The cache size specifies the number of buckets in the active instance list within the EJB container.

To change these settings, click **Servers -> Application Servers -> <ServerName> -> EJB Container -> EJB Cache Settings** (from the Additional Properties pane).

The default values are Cache size=2053 and Cache cleanup interval=3000 milliseconds.

### ***ORB thread pool size***

Method invocations to enterprise beans are only queued for requests coming from remote clients going through the RMI activity service. An example of such a client is an EJB client running in a separate Java Virtual Machine (another address space) from the enterprise bean. In contrast, no queuing occurs if the EJB client (either a servlet or another enterprise bean) is installed in the same JVM that the EJB method runs on and the same thread of execution as the EJB client.

Remote enterprise beans communicate by using the RMI/IIOP protocol. Method invocations initiated over RMI/IIOP are processed by a server-side ORB. The thread pool acts as a queue for incoming requests. However, if a remote method request is issued and there are no more available threads in the thread pool, a new thread is created. After the method request completes, the thread is destroyed. Therefore, when the ORB is used to process remote method requests, the EJB container is an open queue, due to the use of unbounded threads.

Tivoli Performance Viewer can help tune the ORB thread pool size settings. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the Percent Maxed counter of the Object Request Broker submodule of the Thread Pools module. If the value of this counter is consistently in the double digits, then the ORB could be a bottleneck and the number of threads in the pool should be increased.

The degree to which the ORB thread pool value needs to be increased is a function of the number of simultaneous servlets (that is, clients) calling enterprise beans and the duration of each method call. If the method calls are longer or the

applications spend a lot of time in the ORB, consider making the ORB thread pool size equal to the Web container size. If the servlet makes only short-lived or quick calls to the ORB, servlets can potentially reuse the same ORB thread. In this case, the ORB thread pool can be small, perhaps even one-half of the thread pool size setting of the Web container.

The ORB thread pool size is configured from the Administrative Console using these steps:

1. Select **Servers** -> **Application Servers** in the console navigation tree.
2. Select the name of the application server from the list of application servers in the workspace.
3. Select the **ORB Service** entry in the Additional Properties pane of the workspace.
4. Select **Thread Pool** in the Additional Properties pane of the workspace.
5. Use the Maximum Size field to configure the maximum pool size. Note that this only affects the number of threads held in the pool (the actual number of ORB threads can be higher).
6. Save the configuration and restart the affected application server for the change to take effect.

Some additional settings related to the ORB can also be tuned. These are explained in 19.4.12, “Object Request Broker (ORB)” on page 903.

### ***Break CMP enterprise beans into several enterprise bean modules during assembly***

To increase performance, break CMP enterprise beans into several enterprise bean modules during assembly. The load time for hundreds of beans can be improved by distributing the beans across several JAR files and packaging them to an EAR file. This is faster when the administrative server attempts to start the beans, for example 8-10 minutes versus more than one hour when one JAR file is used.

## **Web container**

To route servlet requests from the Web server to the Web containers, a transport queue between the Web server plug-in and each Web container is established. The number of client requests accepted by the container is determined by the Web container thread pool. Connection reuse is another factor that influences the number of concurrent threads that are processed by the Web container.



### ***Thread pool***

The Web container maintains a thread pool to process inbound HTTP(S) requests for resources in the container (that is servlets and JSPs). This is a closed queue, since the thread pool is bounded by the maximum thread size.

Tivoli Performance Viewer can help tune the Web container thread pool size settings. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the Percent Maxed and Active Threads counters of the Web container submodule of the Thread Pools module. If the value of the Percent Maxed counter is consistently in the double digits, then the Web container could be a bottleneck and the number of threads should be increased. On the other hand if the number of active threads are significantly lower than the number of threads in the pool, consider lowering the thread pool size for a performance gain.

**Note:** For Linux systems, the recommended value for the Web container maximum thread pool size used to be 25. However, new Linux kernel versions might support a higher value. The WebSphere performance team is currently working on a list of Linux distributions/kernel versions that support higher values. Please monitor the WebSphere InfoCenter for updates on this.

The Web container thread pool size is configured from the Administrative Console using these steps:

1. Select **Servers** -> **Application Servers** in the console navigation tree.
2. Select the name of the application server from the list of application servers in the workspace.
3. Select the **Web container** entry in the Additional Properties pane of the workspace.
4. Select the **Thread Pool** entry in the Additional Properties pane of the workspace.
5. Use the Maximum Size field to configure the maximum pool size. Note that in contrast to the ORB, the Web container only uses threads from the pool, hence a closed queue. The default value is 50.
6. Save the configuration and restart the affected application server for the change to take effect.

**Important:** Checking the **Growable Thread Pool** box on the Thread Pool Configuration page allows for an automatic increase of the number of threads beyond the maximum size configured for the thread pool. As a result of this, the system can become overloaded because too many threads are allocated.

## **MaxKeepAliveConnections**

The MaxKeepAliveConnections parameter describes the maximum number of concurrent connections to the Web container that are allowed to be kept alive, that is, to be processed in multiple requests. The Web server plug-in keeps connections open to the application server as long as it can. However, if the value of this property is too small, performance is negatively impacted because the Web server plug-in has to open a new connection for each request instead of sending multiple requests through one connection.

The **netstat** command utility can help tune the maximum keep-alive connections setting. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the number of connections in the TIME\_WAIT state to the application server port. If the count of TIME\_WAITs is consistently in the double digits, it might improve performance to raise the maximum keep-alive connections or maximum keep-alive requests properties (described in “MaxKeepAliveRequests” on page 859). Commands for retrieving the count of TIME\_WAITs are shown in Example 19-4. Substitute the port number with the port of the specific application server you want to monitor. Be aware that having both the Web server and application server installed on the same machine would result in a double count of every connection (since the TIME\_WAIT state is listed from both the client side and server side by netstat).

---

### *Example 19-4 Using netstat to determine the time\_wait values*

---

On the Windows platform the chain of commands would be:

```
netstat -na | find /i "time_wait" | find /c "9080"
```

On the Unix platform the chain of commands would look like:

```
netstat -na | grep -i time_wait | grep -c 9080
```

---

The maximum number of keep-alive connections allowed to the Web container is configured from the Administrative Console using these steps:

1. Select **Servers -> Application Servers** in the console navigation tree.
2. Select the name of the application server from the list of application servers in the workspace.
3. Select the **Web Container** entry in the Additional Properties pane of the workspace.
4. Select **HTTP Transports** in the Additional Properties pane of the workspace.
5. Select the host link for which to configure the max keep-alive connections setting in the host column of the HTTP Transports pane in the workspace.
6. Select **Custom properties** in the Additional Properties pane of the workspace.

7. Click the **New** button.
8. Enter `MaxKeepAliveConnections` in the Name field and an integer value in the Value field (the recommended starting value is 90% of the maximum threads in the Web container thread pool).
9. Click **OK**, save the configuration, and restart the affected application server for the change to take effect.

The value should be at least 90% of the maximum number of threads in the Web container thread pool. If it is 100% of the maximum number of threads in the Web container thread pool, all the threads could be consumed by keep-alive connections leaving no threads available to process new connections.

The application server might not accept a new connection under a heavy load if there are too many sockets in `TIME_WAIT` state.

If all client requests are going through the Web server plug-in and there are many `TIME_WAIT` state sockets for the Web container port, the application server is closing connections prematurely, which decreases performance. The application server will close the connection from the Web server plug-in, or from any client, for any of the following reasons:

- ▶ The client request was an HTTP 1.0 request when the Web server plug-in always sends HTTP 1.1 requests.
- ▶ The maximum number of concurrent keep-alive connections was reached.
- ▶ The maximum number of requests for a connection was reached.
- ▶ A timeout occurred while waiting to read the next request or to read the remainder of the current request.

### ***MaxKeepAliveRequests***

The `MaxKeepAliveRequests` is the maximum number of requests allowed on a single keep-alive connection. This parameter can help prevent denial of service attacks when a client tries to hold on to a keep-alive connection. The Web server plug-in keeps connections open to the application server as long as it can, providing optimum performance.

A good starting value for the maximum number of requests allowed is 100. If the application server requests are received from the Web server plug-in only, increase this parameter's value. The `netstat` utility can be used to tune the value of maximum keep-alive requests as described in “`MaxKeepAliveConnections`” on page 858. If the number of connections in the `TIME_WAIT` state is too high, consider raising the maximum keep-alive requests setting.

The maximum number of requests allowed is configured from the Administrative Console using these steps:

1. Select **Servers -> Application Servers** in the console navigation tree.
2. Select the name of the application server from the list of application servers in the workspace.
3. Select the **Web Container** entry in the Additional Properties pane.
4. Select **HTTP Transports** in the Additional Properties pane.
5. Select the host link for which to configure the max keep-alive requests setting in the host column of the HTTP Transports pane in the workspace.
6. Select **Custom properties** in the Additional Properties pane.
7. Click the **New** button.
8. Enter `MaxKeepAliveRequests` in the Name field and an integer value in the Value field (the recommended starting value is 100).
9. Click **OK** to store the property.
10. Save the configuration and restart the affected application server for the change to take effect.

### ***HTTP transport custom properties***

Following is a list of additional custom properties of the HTTP transport provided with the application server. These custom properties are useful for further tuning measures. Like the previous two, `MaxKeepAliveRequests` and `MaxKeepAliveSessions`, these custom properties are not shown in the settings page for an HTTP transport, but have to be set via the **Custom properties** pane in the same way as described above.

► **ConnectionLOutTimeOut**

Specifies the maximum number of seconds to wait when trying to read or process data during a request. The default value is 5 seconds.

► **ConnectionResponseTimeout**

Specifies the maximum number of seconds to wait when trying to read or write data during a response. The default is 300 seconds.

► **MaxConnectBacklog**

Specifies the maximum number of outstanding connect requests that the operating system will buffer while it waits for the application server to accept the connections. If a client attempts to connect when this operating system buffer is full, the connect request will be rejected. The default setting is 511.

Set this value to the number of concurrent connections that you would like to allow. Keep in mind that a single client browser might need to open multiple concurrent connections (perhaps four or five); however, also keep in mind that

increasing this value consumes more kernel resources. The value of this property is specific to each transport.

► **KeepAliveEnabled**

Specifies whether or not to keep connections alive, that is, persistent. The default value is true.

The Web server plug-in keeps connections open to the application server as long as it can, bounded by the following keep alive parameter settings:

– **MaxKeepAliveConnections**

Specifies the maximum number of concurrent keep alive (persistent) connections across all HTTP transports. To make a particular transport close connections after a request, you can set `MaxKeepAliveConnections` to 0 (zero) or you can set `KeepAliveEnabled` to false on that transport. The default is 90% of the maximum number of threads in the Web container thread pool (default maximum thread pool size = 50, which means that the default `MaxKeepAliveConnections` = 45).

– **MaxKeepAliveRequests**

Specifies the maximum number of requests which can be processed on a single keep alive connection. This parameter can help prevent denial-of-service attacks when a client tries to hold on to a keep-alive connection. The default is 100 requests.

– **ConnectionKeepAliveTimeout**

Specifies the maximum number of seconds to wait for the next request on a keep alive connection.

## **Web server**

In order to utilize WebSphere Application Server resources on request processing instead of request queuing, the client requests that arrive at the Web server when the WebSphere Application Server is saturated should be queued in the network. All Web servers have settings for configuring the maximum number of concurrent requests accepted. For information about how to configure this setting, refer to 19.4.8, “The Web server” on page 887. For details the new features of IBM HTTP Server 2.0 and how to use them, refer to “IBM HTTP Server 2.0” on page 894.

### ***Limiting connections from plug-in to Web container***

In WebSphere Application Server V5.1, the plug-in has been enhanced with the *MaxConnections* attribute. The *MaxConnections* value specifies the maximum number of connections permitted concurrently to each application server. When this number of connections is reached, the plug-in automatically skips that application server on establishing new connections, and tries the next available one (see also “*MaxConnections* setting” on page 210).

The MaxConnections attribute works best with Web servers that follow the threading model instead of the process model, and where only one process is started (also called *single process-multiple threads* model). This is true for IBM HTTP Server version 2, which is the recommended Web server for WebSphere Application Server V5.1.

**Important:** IBM HTTP Server v1.3.x follows the process model (in contrary to IBM HTTP Server v2.0.x). With the process model, a new process is created to handle each connection from the application server, and typically, one process handles only one connection to the application server. Therefore, the MaxConnections attribute does not have much of an impact in restricting the number of concurrent requests to the application server.

### ***Determining the Web server maximum concurrency threads setting***

A Web server monitor (see 16.8, “Monitoring the IBM HTTP Server” on page 730) can help tune the maximum concurrent thread processing setting for the Web server. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the number of Web server threads going to the Web container and the number of threads accessing static content locally on the Web server. Set the concurrent requests setting to the sum of the Web container thread pool size and the number of static content requests processed by the Web server. If no static content is served from the Web server, the number of requests should be about equal to the Web container thread pool size (depending on how many clients and Web servers are concurrently accessing the same Web container).

### **Using cluster configurations**

The capability to spread workload among application servers using clustering can be a valuable asset in configuring highly scalable production environments. This is especially true when the application is experiencing bottlenecks that are preventing full CPU utilization of Symmetric Multiprocessing (SMP) servers. When adjusting the WebSphere system queues in clustered configurations, remember that when a server is added to a cluster, the server downstream receives twice the load. This is illustrated in Figure 19-7 on page 863.

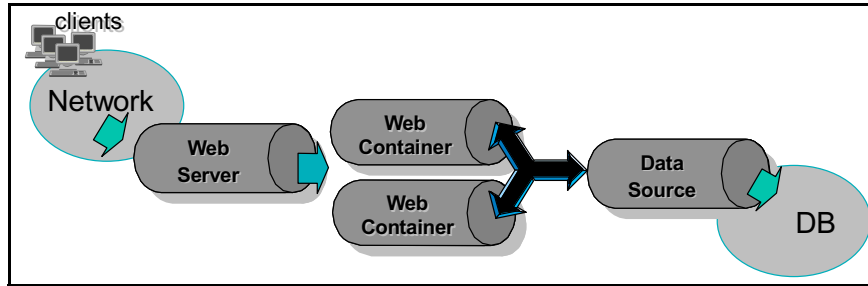


Figure 19-7 Clustering and queuing

Two Web containers within a cluster are located between a Web server and a data source. It is assumed the Web server, Web container, and data source (but not the database) are all running on a single SMP server. Given these constraints, the following queue considerations need to be made:

- ▶ Web server queue settings can be doubled to ensure ample work is distributed to each Web container.
- ▶ Web container thread pools can be reduced to avoid saturating a system resource such as CPU or another resource that the servlets are using.
- ▶ The data source pools can be reduced to avoid saturating the database server.
- ▶ Java heap parameters can be reduced for each instance of the application server. For versions of the JVM shipped with WebSphere Application Server, it is crucial that the heap from all JVMs remain in physical memory. Therefore, if a cluster of four JVMs are running on a system, enough physical memory must be available for all four heaps.

### 19.4.5 Application assembly performance checklist

Application assembly tools are used to assemble J2EE components and modules into J2EE applications. Generally, this consists of defining application components and their attributes including enterprise beans, servlets and resource references. Many of these application configuration settings and attributes play an important role in the runtime performance of the deployed application. The most important parameters and advice for finding optimal settings are:

- ▶ Enterprise bean modules:
  - Entity EJBs - Bean cache
  - Method extensions - Isolation level
  - Method extensions - Access intent
  - Container transactions

- Web modules:
  - Web application - Distributable
  - Web application - Reload interval
  - Web application - Reload enabled
  - Web application - Web components - Load on startup

## Enterprise bean modules

This section explains the enterprise bean module parameters mentioned above in detail.

**Note:** Although WebSphere Application Server 5.1 also supports EJB 2.0, the following information refers to EJB 1.1 settings.

### ***Entity EJBs - Bean cache***

WebSphere Application Server provides significant flexibility in the management of database data with Entity EJBs. The Entity EJBs Activate at and Load at configuration settings specify how and when to load and cache data from the corresponding database row data of an enterprise bean. These configuration settings provide the capability to specify enterprise bean caching Options A, B or C, as specified in the EJB 1.1 specification.

Option A provides maximum enterprise bean performance by caching database data outside of the transaction scope. Generally, Option A is only applicable where the EJB container has exclusive access to the given database. Otherwise, data integrity is compromised. Option B provides more aggressive caching of Entity EJB object instances, which can result in improved performance over Option C, but also results in greater memory usage. Option C is the most common real-world configuration for Entity EJBs.

#### ► Bean cache - Activate at

This setting specifies the point at which an enterprise bean is activated and placed in the cache. Removal from the cache and passivation are also governed by this setting. Valid values are Once and Transaction. Once indicates that the bean is activated when it is first accessed in the server process, and passivated (and removed from the cache) at the discretion of the container, for example when the cache becomes full. Transaction indicates that the bean is activated at the start of a transaction and passivated (and removed from the cache) at the end of the transaction. The default value is Transaction.

#### ► Bean cache - Load at

This setting specifies when the bean loads its state from the database. The value of this property implies whether the container has exclusive or shared access to the database. Valid values are Activation and Transaction.



Activation indicates the bean is loaded when it is activated and implies that the container has exclusive access to the database. Transaction indicates that the bean is loaded at the start of a transaction and implies that the container has shared access to the database. The default is Transaction.

The settings of the Activate at and Load at properties govern which commit options are used.

- For Option A (exclusive database access), use Activate at = Once and Load at = Activation.

This option reduces database input/output by avoiding calls to the `ejbLoad` function, but serializes all transactions accessing the bean instance. Option A can increase memory usage by maintaining more objects in the cache, but can provide better response time if bean instances are not generally accessed concurrently by multiple transactions.

**Note:** When using WebSphere Network Deployment and workload management is enabled, Option A cannot be used. You must use settings that result in the use of Options B or C.

- For Option B (shared database access), use Activate at = Once and Load at = Transaction.

Option B can increase memory usage by maintaining more objects in the cache. However, because each transaction creates its own copy of an object, there can be multiple copies of an instance in memory at any given time (one per transaction), requiring the database be accessed at each transaction. If an enterprise bean contains a significant number of calls to the `ejbActivate` function, using Option B can be beneficial because the required object is already in the cache. Otherwise, this option does not provide significant benefit over Option A.

- For Option C (shared database access), use Activate at = Transaction and Load at = Transaction.

This option can reduce memory usage by maintaining fewer objects in the cache. However, there can be multiple copies of an instance in memory at any given time (one per transaction). This option can reduce transaction contention for enterprise bean instances that are accessed concurrently but not updated.

### ***Method extensions - Isolation level***

WebSphere Application Server enterprise bean method extensions provide settings to specify the level of transactional isolation used when accessing data.

Isolation level settings specify various degrees of runtime data integrity provided by the corresponding database. First, choose a setting that meets data integrity requirements for the given application and specific database characteristics.

The valid values are:

- ▶ Serializable
- ▶ Repeatable read
- ▶ Read committed
- ▶ Read uncommitted

Isolation level also plays an important role in performance. Higher isolation levels reduce performance by increasing row locking and database overhead while reducing data access concurrency. Various databases provide different behavior with respect to the isolation settings. In general, Repeatable read is an appropriate setting for DB2 databases. Read committed is generally used for Oracle. Oracle does not support Repeatable read and will translate this setting to the highest isolation level of Serializable.

The Isolation level can be specified at the bean or method level. Therefore, it is possible to configure different isolation level settings for various methods. This is an advantage when some methods require higher isolation than others, and can be used to achieve maximum performance while maintaining integrity requirements. However, isolation cannot change between method calls within a single enterprise bean transaction. A runtime exception will be thrown in this case.

The following section describes the four isolation levels:

- ▶ Serializable

This level prohibits the following types of reads:

- *Dirty reads*: A transaction reads a database row containing uncommitted changes from a second transaction.
- *Nonrepeatable reads*: One transaction reads a row, a second transaction changes the same row, and the first transaction rereads the row and gets a different value.
- *Phantom reads*: One transaction reads all rows that satisfy an SQL WHERE condition, a second transaction inserts a row that also satisfies the WHERE condition, and the first transaction applies the same WHERE condition and gets the row inserted by the second transaction.

- ▶ Repeatable read

This level prohibits dirty reads and nonrepeatable reads, but it allows phantom reads.

- Read committed

This level prohibits dirty reads, but allows nonrepeatable reads and phantom reads.

- Read uncommitted

This level allows dirty reads, nonrepeatable reads, and phantom reads.

The container uses the transaction isolation level attribute as follows:

- Session beans and entity beans with bean-managed persistence (BMP):

For each database connection used by the bean, the container sets the transaction isolation level at the start of each transaction unless the bean explicitly sets the isolation level on the connection.

- Entity beans with container-managed persistence (CMP):

The container generates database access code that implements the specified isolation level.

### ***Method extensions - Access intent***

WebSphere Application Server enterprise bean method extensions provide settings to specify individual enterprise bean methods as read-only. This setting denotes whether the method can update entity attribute data (or invoke other methods that can update data in the same transaction).

**Note:** This setting is applicable only for EJB 1.x-compliant beans, that is:

- EJB 1.x compliant entity beans
- Enterprise beans with CMP Version 1.x that are packaged in EJB 2.x-compliant modules.

To specify the access intent for EJB 2.x-compliant beans, select an access intent policy.

By default, all enterprise bean methods are assumed to be "update" methods. This results in EJB Entity data always being persisted back to the database at the close of the enterprise bean transaction. Marking enterprise methods that do not update entity attributes as Access Intent Read, provides a significant performance improvement by allowing the WebSphere Application Server EJB container to skip the unnecessary database update.

A behavior for "finder" methods for CMP Entity EJBs is available. By default, WebSphere Application Server will invoke a Select for Update query for CMP enterprise bean finder methods such as `findByPrimaryKey`. This exclusively locks the database row for the duration of the enterprise bean transaction. However, if the enterprise bean finder method has been marked as Access Intent Read, the

container will not issue the For Update on the select, resulting in only a read lock on the database row.

### ***Container transactions***

The container transaction setting specifies how the container manages transaction scopes when delegating invocation to the enterprise bean individual business method. The legal values are:

- ▶ Never
- ▶ Mandatory
- ▶ Requires New
- ▶ Required
- ▶ Supports
- ▶ Not Supported
- ▶ Bean Managed

The container transactions attribute can be specified individually for one or more enterprise bean methods. Enterprise bean methods not requiring transactional behavior can be configured as Supports to reduce container transaction management overhead.

The following section describes the legal values in detail:

- ▶ Never

This legal value directs the container to invoke bean methods without a transaction context. If the client invokes a bean method from within a transaction context, the container throws the `java.rmi.RemoteException` exception.

If the client invokes a bean method from outside a transaction context, the container behaves in the same way as if the Not Supported transaction attribute was set. The client must call the method without a transaction context.

- ▶ Mandatory

This legal value directs the container to always invoke the bean method within the transaction context associated with the client. If the client attempts to invoke the bean method without a transaction context, the container throws the `javax.transaction.TransactionRequiredException` exception to the client. The transaction context is passed to any enterprise bean object or resource accessed by an enterprise bean method.

Enterprise bean clients that access these entity beans must do so within an existing transaction. For other enterprise beans, the enterprise bean or bean method must implement the Bean Managed value or use the Required or Requires New value. For non-enterprise bean EJB clients, the client must invoke a transaction by using the `javax.transaction.UserTransaction` interface.

► **Requires New**

This legal value directs the container to always invoke the bean method within a new transaction context, regardless of whether the client invokes the method within or outside a transaction context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

► **Required**

This legal value directs the container to invoke the bean method within a transaction context. If a client invokes a bean method from within a transaction context, the container invokes the bean method within the client transaction context. If a client invokes a bean method outside a transaction context, the container creates a new transaction context and invokes the bean method from within that context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

► **Supports**

This legal value directs the container to invoke the bean method within a transaction context if the client invokes the bean method within a transaction. If the client invokes the bean method without a transaction context, the container invokes the bean method without a transaction context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

► **Not Supported**

This legal value directs the container to invoke bean methods without a transaction context. If a client invokes a bean method from within a transaction context, the container suspends the association between the transaction and the current thread before invoking the method on the enterprise bean instance. The container then resumes the suspended association when the method invocation returns. The suspended transaction context is not passed to any enterprise bean objects or resources that are used by this bean method.

► **Bean Managed**

This value notifies the container that the bean class directly handles transaction demarcation. This property can be specified only for session beans and (in EJB 2.0 implementations only) for message-driven beans, not for individual bean methods.

## **Web modules**

This section explains the parameters that can be set for Web modules.

### ***Web application - Distributable***

The distributable flag for J2EE Web applications specifies that the Web application is programmed to be deployed in a distributed servlet container.

**Important:** Web applications should be marked as Distributable if, and only if, they will be deployed in a WebSphere Application Server clustered environment.

### ***Web application - Reload interval***

The reload interval specifies a time interval, in seconds, in which the Web application's file system is scanned for updated files, such as servlet class files or JSPs.

The Reload interval can be defined at different levels for various application components. Generally, the reload interval specifies the time the application server will wait between checks to see if dependent files have been updated and need to be reloaded. Checking file system time stamps is an expensive operation and should be reduced. The default is 0 (zero). Setting this to a value of 3 seconds is good for a test environment, because the Web site can be updated without restarting the application server. In production environments, checking a few times a day is a more common setting.

### ***Web application - Reloading enabled***

This specifies whether file reloading is enabled. The default is false.

### ***Web application - Web components - Load on startup***

Indicates whether a servlet is to be loaded at the startup of the Web application. The default is false.

Many servlets perform resource allocation and other up-front processing in the servlet `init()` method. These initialization routines can be costly at runtime. By specifying Load on startup for these servlets, processing takes place when the application server is started. This avoids runtime delays, which can be encountered on a servlet's initial access.

## **19.4.6 Java tuning**

The following section focuses on tuning Java memory. Enterprise applications written in Java involve complex object relationships and utilize large numbers of objects. Although Java automatically manages memory associated with an object's *life cycle*, understanding the application's usage patterns for objects is important. In particular, ensure the following:

- The application is not over-utilizing objects

- ▶ The application is not leaking objects (that is, memory)
- ▶ The Java heap parameters are set to handle the use of objects

Understanding the effect of garbage collection is necessary to apply these management techniques.

## Garbage collection basics

Garbage collection algorithms, like JVMs, have evolved and become more and more complex to understand. Knowledge of how the Garbage Collector (GC) works is necessary for designing and tuning Java applications and application servers. Following is a broad, somewhat simplified, overview of the Mark - Sweep - Compact (MSC) garbage collection technique implemented by IBM JVMs. For an in-depth study of additional, state-of-the-art heap management and garbage collection techniques, refer to the articles mentioned in “Additional JVM and garbage collection related resources” on page 879.

The Garbage Collector allocates areas of storage inside the Java heap, where objects, arrays, and classes are stored. An allocated object is considered *live* when there exists at least one reference to it, that means, it is used by someone, commonly another object. Thus the object is also considered *reachable*. When this object is no longer used by anyone, all references should have been removed, it is now considered *garbage*, and its allocated storage area should be reclaimed for reuse. This task is performed by the Garbage Collector.

When the JVM is unable to allocate an object from the current Java heap because of lack of free, contiguous space, a memory allocation fault occurs (allocation failure) and the Garbage Collector is invoked. (The GC can also be invoked by a specific function call: `System.gc()`. However, when `System.gc()` is invoked, the JVM can simply 'take it under advisement' and choose to defer the GC operation until later if the JVM has more pressing needs to attend to.) The first task of the GC is to make sure to acquire all locks required for garbage collection, and then stops all the other threads. Because of this garbage collection is also referred to as *stop-the-world* (STW) collection. Garbage collection will then take place in three phases: mark, sweep, and optionally compact.

### **Mark phase**

In the mark phase, all *reachable* objects that are referenced either directly by the JVM, for example through threads stacks, or in turn by other objects, will be identified. Everything else that is not marked is considered garbage.

### **Sweep phase**

All allocated objects that are not marked are swept away, that is, the space used by them is reclaimed.

### ***Compaction phase***

When the garbage has been removed from the heap, the GC can consider compacting the heap, which is typically riddled with holes caused by the freed objects by now. When there is no chunk of memory available big enough to satisfy an allocation request after garbage collection, the heap has to be compacted. Because heap compaction means moving objects around and updating all references to them, it is extremely costly in terms of time, and the GC tries to avoid it if possible. Modern JVM implementations try to avoid heap compaction by focusing on optimizing object placement in the heap.

**Note:** Do not confuse the Java heap with the native (or system) heap! The native heap is never garbage collected; it is used by the JVM process and stores all the objects that the JVM itself needs during its entire lifetime. The native heap is typically much smaller than the Java heap.

### ***Heap expansion and shrinkage***

Heap expansion will occur after garbage collection if the ratio of free to total heap size falls below the value specified by the `-Xminf` parameter. The default is 0.3 (or 30%).

Heap shrinkage will occur after garbage collection if the ratio of free to total heap size exceeds the value specified by the `-Xmaxf` parameter. The default is 0.6 (or 60%). The amount of expansion is governed by the minimum expansion size, set by the `-Xmine` parameter, and the maximum expansion size, defined by `-Xmaxe`. The defaults for `-Xmine` are 1MB, for `-Xmaxe` 0, which is equal to unlimited. These parameters do not have any effect on a fixed-size heap, where the `-Xms` and `-Xmx` values are equal.

### ***Parallel versus concurrent operation***

Recent releases of JVMs implement multiple helper threads that run in parallel on a multi-processor machine during the mark and sweep phases. These threads are asleep during normal operation, and only during garbage collection the work is divided between the main GC thread and his helper threads, using all processors simultaneously. *Parallel mark* mode is enabled by default since IBM JVM versions 1.3.0, while *parallel sweep* is enabled by default since version 1.3.1. As mentioned before, garbage collection is basically a stop-the-world operation. This is not entirely true: IBM JVM 1.3.1 and higher also know an optional *concurrent mark* mode, where a background thread is started by the JVM, and some of the work of the mark phase is done concurrently while all application threads are active. Thus the STW pause will be reduced when garbage collection finally occurs. Concurrent mark is disabled by default, and can be activated using the `-Xgcpolicy:optavgpause` parameter.



**Note:** In some cases, concurrent mark may reduce the throughput of an application. It is recommended you compare the application performance with and without concurrent mark, using identical loads to measure the effect on application performance.

In addition, IBM JVM 1.4 knows an *incremental compaction mode* which parallelizes the compaction phase. For an introductory explanation of parallel and concurrent modes, refer to the developerWorks article *Fine-tuning Java garbage collection performance* by Sumit Chawla found at:

<http://www.ibm.com/developerworks/ibm/library/i-gctroub/>

or the IBM JVM Diagnostics Guides (see “Additional JVM and garbage collection related resources” on page 879) for an in-depth discussion of these features.

## The garbage collection bottleneck

Examining Java garbage collection can give insight into how the application is utilizing memory. Garbage collection is a Java strength. By taking the burden of memory management away from the application writer, Java applications are more robust than applications written in languages that do not provide garbage collection. This robustness applies as long as the application is not abusing objects. Garbage collection normally consumes anywhere from 5 to 20% of the total execution time of a properly functioning application. If not managed, garbage collection can be one of the biggest bottlenecks for an application, especially when running on SMP server machines.

## The garbage collection gauge

Garbage collection can be used to evaluate application performance health. By monitoring garbage collection during the execution of a fixed workload, users gain insight as to whether the application is over-utilizing objects. Garbage collection can even be used to detect the presence of memory leaks.

Use the garbage collection and heap statistics in Tivoli Performance Viewer (see 16.3, “Using Tivoli Performance Viewer” on page 705) to evaluate application performance health. Mind that you have to enable the JVMPI facility (see “Performance data provided by JVMPI” on page 702) to get detailed garbage collection statistics. By monitoring garbage collection, memory leaks and overly used objects can be detected.

For this type of investigation, set the minimum and maximum heap sizes to the same value. Choose a representative, repetitive workload that matches production usage as closely as possible, user errors included. To ensure meaningful statistics, run the fixed workload until the state of the application is steady. Reaching this state usually takes several minutes.

## Detecting over-utilization of objects

To see if the application is overusing objects, look in Tivoli Performance Viewer at the counters for the JVMPI profiler. The average time between garbage collection calls should be 5 to 6 times the average duration of a single garbage collection. If not, the application is spending more than 15% of its time in garbage collection. Also, look at the numbers of freed, allocated, and moved objects.

If the information indicates a garbage collection bottleneck, there are two ways to clear the bottleneck. The most cost-effective way to optimize the application is to implement object caches and pools. Use a Java profiler to determine which objects to target. If the application cannot be optimized, adding memory, processors and application clusters might help. Additional memory allows each application server in a cluster to maintain a reasonable heap size. Additional processors allow the cluster members to run in parallel.

## Detecting memory leaks

Memory leaks in Java are a dangerous contributor to garbage collection bottlenecks. Memory leaks are more damaging than memory overuse, because a memory leak ultimately leads to system instability. Over time, garbage collection occurs more frequently until finally the heap is exhausted and Java fails with a fatal Out of Memory exception. Memory leaks occur when an unneeded object has references that are never deleted. This most commonly occurs in collection classes, such as Hashtable, because the table itself always has a reference to the object, even after real references have been deleted.

High workload often causes many applications to crash immediately after being deployed in the production environment. This situation is especially true for leaking applications where the high workload accelerates the magnification of the leakage and a memory allocation failure occurs.

Memory leak testing relates to magnifying numbers. Memory leaks are measured in terms of the amount of bytes or kilobytes that cannot be garbage collected. The delicate task is to differentiate these amounts from the expected sizes of useful and unusable memory. This task is achieved more easily if the numbers are magnified, resulting in larger gaps and easy identification of inconsistencies. The following is a list of important conclusions about memory leaks:

- Long-running test

Memory leak problems are manifested only after a period of time. Therefore, memory leaks are usually found during long-running tests. Short runs can lead to false alarms. One of the problems in Java is whether to say that a memory leak is occurring when memory usage has seemingly increased either abruptly or monotonically in a given period. These kind of increases can be valid, and the objects created can be referenced at a much later time.

In other words, what method is used to differentiate the delayed use of objects from completely unused objects? Running applications over a long period of time will get a higher confidence for whether the delayed use of objects is actually occurring. Because of this, memory leak testing cannot be integrated with some other types of tests, such as functional testing, that occur earlier in the process. However, tests such as stress or durability tests can be integrated.

- System test

Some memory leak problems occur only when different components of a big project are combined and executed. Interfaces between components can produce known or unknown side effects. System test is a good opportunity to make these conditions happen.

- Repetitive test

In many cases, memory leak problems occur by successive repetitions of the same test case. The goal of memory leak testing is to establish a big gap between unusable memory and used memory in terms of their relative sizes. By repeating the same scenario over and over again, the gap is multiplied in a very progressive way. This testing helps if the amount of leaks caused by an execution of a test case is so minimal that it could hardly be noticed in one run.

Repetitive tests can be used at the system level or module level. The advantage with modular testing is better control. When a module is designed to keep the private module without creating external side effects such as memory usage, testing for memory leaks can be much easier. First, the memory usage before running the module is recorded. Then, a fixed set of test cases is run repeatedly. At the end of the test run, the current memory usage is recorded and checked for significant changes. Remember, garbage collection must be forced when recording the actual memory usage by inserting `System.gc()` in the module where garbage collection should occur or by using a profiling tool to force the event to occur.

- Concurrency test

Some memory leak problems can occur only when there are several threads running in the application. Unfortunately, synchronization points are very susceptible to producing memory leaks because of the added complication in the program logic. Careless programming can lead to references being kept or unreleased. The incident of memory leaks is often facilitated or accelerated

by increased concurrency in the system. The most common way to increase concurrency is to increase the number of clients in the test driver.

Consider the following when choosing which test cases to use for memory leak testing:

- ▶ A good test case exercises areas of the application where objects are created. Most of the time, knowledge of the application is required. A description of the scenario can suggest creation of data spaces, such as adding a new record, creating an HTTP session, performing a transaction and searching a record.
- ▶ Look at areas where collections of objects are being used. Typically, memory leaks are composed of objects of the same class. Also, collection classes such as Vector and Hashtable are common places where references to objects are implicitly stored by calling corresponding insertion methods. For example, the `get` method of a Hashtable object does not remove its reference to the object being retrieved.

Tivoli Performance Viewer helps to find memory leaks. For best results, repeat experiments with increasing duration, such as 1000, 2000, and 4000-page requests. The Tivoli Performance Viewer graph of used memory should have a sawtooth shape. Each drop on the graph corresponds to a garbage collection. There is a memory leak if one of the following occurs:

- ▶ The amount of memory used immediately after each garbage collection increases significantly. The sawtooth pattern will look more like a staircase.
- ▶ The sawtooth pattern has an irregular shape.

Also, look at the difference between the number of objects allocated and the number of objects freed. If the gap between the two increases over time, there is a memory leak.

If heap consumption indicates a possible leak during a heavy workload (the application server is consistently near 100% CPU utilization), yet the heap appears to recover during a subsequent lighter or near-idle workload, this is an indication of heap fragmentation. Heap fragmentation can occur when the JVM is able to free sufficient objects to satisfy memory allocation requests during garbage collection cycles, but the JVM does not have the time to compact small free memory areas in the heap into larger contiguous spaces.

Another form of heap fragmentation occurs when small objects (less than 512 bytes) are freed. The objects are freed, but the storage is not recovered, resulting in memory fragmentation.

Heap fragmentation can be avoided by turning on the `-Xcompactgc` flag in the JVM advanced settings command-line arguments. The `-Xcompactgc` ensures that

each garbage collection cycle eliminates fragmentation, but this setting has a small performance penalty.

## Java heap parameters

The Java heap parameters also influence the behavior of garbage collection. Increasing the heap size allows more objects to be created. Because a large heap takes longer to fill, the application runs longer before a garbage collection occurs. However, a larger heap also takes longer to compact and causes garbage collection to take longer.

For performance analysis, the initial and maximum heap sizes should be equal, as this will eliminate heap growing and shrinking delays. Equating initial with maximum heap size without previous heap size tuning will in most cases create an inefficiently used heap: when it is sized too big, the heap is not used by the application entirely and thus memory resources are wasted.

When tuning a production system where the working set size of the Java application is not understood, a good starting value is to let the initial heap size be 25% of the maximum heap size. The JVM will then try to adapt the size of the heap to the working set size of the application.

Run a series of test experiments that vary the Java heap settings. For example, run experiments with 128 MB, 192 MB, 256 MB, and 320 MB. During each experiment, monitor the total memory usage. If the heap is expanded too aggressively, paging can occur (use the `vmstat` command or the Windows NT or Windows 2000 Performance Monitor to check for paging). If paging occurs, reduce the size of the heap or add more memory to the system.

**Important:** Make sure that the heap never pages, as that would introduce a enormous performance loss.

When all test runs are finished, compare the following statistics:

- ▶ Number of garbage collection calls
- ▶ Average duration of a single garbage collection call
- ▶ Average time between calls
- ▶ Ratio between the average length of a single garbage collection call and the average time between calls

If the application is not over-utilizing objects and has no memory leaks, a state of steady memory utilization is reached. Garbage collection also occurs less frequently and for shorter durations.

If the heap free time settles at 85% or more, consider decreasing the maximum heap size values, because the application server and the application are under-utilizing the memory allocated for heap.

The best result for the average time between garbage collections is at least five to six times the average duration of a single garbage collection. If you do not achieve this number, the JVM is spending more than 15% of its time in garbage collection. Finding this point might require trial and error. A longer interval between garbage collection cycles can result in longer GC runs, while very short intervals can be inefficient.

### ***Heap thrashing***

Avoid heap thrashing at all costs. It is caused by a heap that is barely large enough to avoid expansion but not large enough to satisfy future allocation failures. Usually a garbage collection cycle frees up enough space for not only the current allocation failure but a substantial number of future allocation requests. But when heap is getting thrashed, each garbage collection cycle frees up barely enough heap space to satisfy just the current allocation failure. The result is that the next allocation request leads to another garbage collection cycle, and so on. This scenario can also occur due to lots of short-lived objects.

## **Tuning the IBM JVM**

Every JVM generally offers a whole set of tuning parameters affecting the performance of application servers and applications. Since JVM tuning is a wide and complex topic, this section will present an introduction into garbage collection analysis and tuning in reference to the IBM JVM and provide a series of links and resources for further study.

### ***IBM Java 2 SDK 1.4.1***

WebSphere Application Server V5.1 has been upgraded to support IBM Java 2 SDK version 1.4.1 on all platforms, although it still does not yet exploit many features of version 1.4. However, user applications do have access to all new features. More information about the SDK 1.4.1 enhancements can be found at

[http://java.sun.com/products/archive/j2se/1.4.1\\_07/changes.html](http://java.sun.com/products/archive/j2se/1.4.1_07/changes.html)

### ***Analyzing verbosegc output***

Using verbosegc as the next step after using Tivoli Performance Viewer, is a very good way to see what is going on with garbage collection. Verbosegc mode is enabled by the -verbosegc command-line option, and output is directed to the file native\_stderr.log. This information can then be used to tune the heap size or diagnose problems. For excellent resources and illustrated examples of verbosegc analysis, refer to the following resources:

- ▶ *Sensible Sanitation: Understanding the IBM Java Garbage Collection Part 3: verbosegc and command-line parameters*

<http://www.ibm.com/developerworks/library/i-garbage3.html>

- ▶ *Fine-tuning Java garbage collection performance*

<http://www.ibm.com/developerworks/ibm/library/i-gctroub/>

### ***Common problems***

Following is a short list of common problems and suggested solutions:

- ▶ The GC frequency is too high until the heap reaches a steady state.  
Use verbosegc to determine the size of the heap at a steady state and set -Xms to this value.
- ▶ The heap is fully expanded and the occupancy level is greater than 70%.  
Increase the -Xmx value so the heap is not more than 70% occupied.
- ▶ At 70% occupancy the frequency of GCs is too great.  
Change the setting of -Xminf. The default is 0.3, which will try to maintain 30% free space by expanding the heap. A setting of 0.4 increases this free space target to 40%, reducing the frequency of GCs.
- ▶ Pause times are too long.  
Try using -Xgcpolicy:optavgpause (introduced in 1.3.1), which reduces pause times and makes them more consistent as the heap occupancy rises. There is a cost to pay in throughput. This cost varies and will be about 5%.

### ***Additional JVM and garbage collection related resources***

- ▶ *Garbage collection in the 1.4.1 JVM* available from developerWorks at  
<http://www.ibm.com/developerworks/java/library/j-jtp11253/>
- ▶ *A brief history of garbage collection*, developerWorks  
<http://www.ibm.com/developerworks/java/library/j-jtp10283/>
- ▶ *Sensible Sanitation: Understanding the IBM Java Garbage Collection, Parts 1 and 2*, developerWorks  
<http://www.ibm.com/developerworks/ibm/library/i-garbage1/>  
<http://www.ibm.com/developerworks/ibm/library/i-garbage2/>
- ▶ IBM JVM Diagnostics Guides from developerWorks  
<http://www.ibm.com/developerworks/java/jdk/diagnosis/>
- ▶ *Fine-tuning Java garbage collection performance* from developerWorks  
<http://www.ibm.com/developerworks/ibm/library/i-gctroub/>

- ▶ *Mark that trash - Incremental compaction in the IBM JDK Garbage Collector*  
<http://www.ibm.com/developerworks/ibm/library/i-incrcomp/>

## **Tuning the Sun JVM**

The JVM offers several tuning parameters affecting the performance of WebSphere Application Servers and application performance.

### ***Sun JDK 1.3 HotSpot -server warm-up***

The HotSpot JVM introduces adaptive JVM technology containing algorithms for optimizing byte code execution over time. The JVM runs in two modes, -server and -client. Performance can be significantly enhanced if running in -server mode and a sufficient amount of time is allowed for a HotSpot JVM to warm up by performing continuous execution of byte code.

In most cases, -server mode should be run. This produces more efficient runtime execution over extended periods. The -client option can be used if a faster startup time and smaller memory footprint are preferred, at the cost of lower extended performance.

The -server option is enabled by default and is the recommended value. Follow these steps to change the -client or -server mode:

- ▶ Select **Servers -> Application Servers**.
- ▶ Select the application server you want to tune.
- ▶ Select the link **Process Definition** under Additional Properties.
- ▶ Select **Java Virtual Machine** under Additional Properties.
- ▶ Select **Custom Properties** under Additional Properties.
- ▶ Click **New** and enter HotSpotOption in the Name field. Enter -client or -server in the Value field.
- ▶ Restart the application server.

### ***Sun JDK 1.3 HotSpot new generation pool size***

Most garbage collection algorithms iterate every object in the heap to determine which objects to free. The HotSpot JVM introduces generation garbage collection, which makes use of separate memory pools to contain objects of different ages. These pools can be garbage collected independently from one another. The sizes of these memory pools can be adjusted. Extra work can be avoided by sizing the memory pools so that short-lived objects will never live through more than one garbage collection cycle.



If garbage collection has become a bottleneck, try customizing the generation pool settings. The default values are NewSize=2m, MaxNewSize=32m.

- ▶ Select **Servers -> Application Servers**.
- ▶ Select the name of the application server you want to tune.
- ▶ Select the **Process Definition** link under Additional Properties.
- ▶ Select **Java Virtual Machine** under Additional Properties.
- ▶ Enter the following values in the Generic JVM Arguments field:
  - XX:NewSize (lower bound)
  - XX:MaxNewSize (upper bound)
- ▶ Apply and save your changes, then restart the application server.

It is recommended that you set the new generation pool size between 25 to 50% of the total heap size.

### ***Miscellaneous JVM settings***

Following are several generic JVM setting recommendations that apply to most or all existing JVMs.

### ***Just In Time (JIT) compiler***

The Just In Time (JIT) compiler can significantly affect performance. If you disable the JIT compiler, throughput decreases noticeably. Therefore, for performance reasons, keep JIT enabled.

To determine the setting of this parameter:

- ▶ Select **Servers -> Application Servers**.
- ▶ Select the application server you want to tune.
- ▶ Select the link **Process Definition** under Additional Properties.
- ▶ Select **Java Virtual Machine** under Additional Properties.
- ▶ Check the setting of the Disable JIT check box.
- ▶ If changes are made, save them and restart the application server.

### ***Heap size settings***

These parameters can set the maximum and initial heap sizes for the JVM.

In general, increasing the size of the Java heap improves throughput until the heap no longer resides in physical memory. After the heap begins swapping to disk, Java performance drastically suffers. Therefore, the maximum heap size needs to be low enough to contain the heap within physical memory.

The physical memory usage must be shared between the JVM and other applications running on the system, such as the database. For assurance, use a smaller heap, for example 64 MB, on machines with less memory.

Try a maximum heap of 128 MB on a smaller machine, that is, less than 1 GB of physical memory. Use 256 MB for systems with 2 GB memory, and 512 MB for larger systems. The starting point depends on the application.

If performance runs are being conducted and highly repeatable results are needed, set the initial and maximum sizes to the same value. This setting eliminates any heap growth during the run. For production systems where the working set size of the Java applications is not well understood, an initial setting of one-fourth the maximum setting is a good starting value. The JVM will then try to adapt the size of the heap to the working set of the Java application.

- ▶ Select **Servers -> Application Servers**.
- ▶ Select the application server you want to tune.
- ▶ Select **Process Definition** under Additional Properties.
- ▶ Select **Java Virtual Machine** under Additional Properties.
- ▶ In the General Properties configuration, enter values for Initial Heap Size and Maximum Heap Size fields.
- ▶ Apply and save your changes, then restart the application server.

### ***Class garbage collection***

Disabling class garbage collection enables more class reuse, which, in some cases, has resulted in small performance improvements.

In most cases, run with class garbage collection turned on. This is the default.

To disable class garbage collection, enter the value `-Xnoclassgc` in the Generic JVM Arguments field of the application servers' JVM configuration. To do this:

- ▶ Select **Servers -> Application Servers**.
- ▶ Select the application server you want to tune.
- ▶ Select the link **Process Definition** under Additional Properties.
- ▶ Select the link **Java Virtual Machine** under Additional Properties.
- ▶ Enter the value `-Xnoclassgc` in the Generic JVM Arguments field.
- ▶ Apply and save your changes. Restart the application server.

## 19.4.7 Operating system tuning

This section provides some basic information about operating system tuning parameters for AIX, Sun Solaris, and Windows NT/2000. For more details refer to the WebSphere InfoCenter. Expand **Tuning -> System**, then select **Tuning operating systems**.

Over time, more information will be added to the InfoCenter, for example HP-UX related tuning information. Therefore, for the latest information, always check the InfoCenter in addition to this redbook.

### **AIX**

There are many AIX operating system settings to consider that are not within the scope of this redbook. Some of the settings you can adjust are:

- ▶ Adapter transmit and receive queue
- ▶ TCP/IP socket buffer
- ▶ IP protocol mbuf pool performance
- ▶ Update file descriptors
- ▶ Update the scheduler

However, two important settings are outlined in the next sections.

### ***AIX with DB2***

Separating your DB2 log files from the physical database files can boost performance. You can also separate the logging and the database files from the drive containing the Journaled File System (JFS) service. AIX uses specific volume groups and file systems for the JFS logging.

The AIX **filemon** utility is used to view all file system input and output, and to strategically select the file system for the DB2 logs. The default location for the files is `/home/<db2_instance>/<db2_instance>/NODExx/SQLyy/SQLLOGDIR/`.

To change the location of the files, at a DB2 command prompt, issue the following command:

```
db2 update db cfg for [database_name] using newlogpath  
[fully_qualified_path]
```

It is recommended that you move the logs to a separate disk when your application shows more than a 20% I/O wait time.

### ***AIX file descriptors (ulimit)***

Specifies the number of open files permitted.

When should you try adjusting this value? The default setting is typically sufficient for most applications. If the value set for this parameter is too low, a Memory allocation error is displayed.

Check the UNIX reference pages on `ulimit` for the syntax for different shells. For the KornShell shell (`ksh`), to set `ulimit` to 2000, issue the following command:

```
ulimit -n 2000
```

Use `smit` (or `smitty`) to permanently set this value for a user.

Use the command `ulimit -a` to display the current values for all limitations on system resources. The default setting is 2000, which is also the recommended value.

## HP-UX 11i

Some HP-UX 11i settings can be modified to significantly improve WebSphere Application Server performance.

### ***TCP\_CONN\_REQUEST\_MAX***

When high connection rates occur, a large backlog of TCP/IP connection requests build up and client connections are dropped. Adjust this setting when clients start to timeout after waiting to connect. This situation can be verified by issuing the `netstat -p tcp` command. Look for the following value: connect requests dropped due to full queue. In most cases the default (4096) should suffice. Consider adjusting to 8192 if the default proves inadequate. Set this parameter by using the `ndd -set /dev/tcp tcp_conn_request_max` command.

### ***Java virtual machine virtual page size***

Setting the Java virtual machine instruction and data page sizes to 64 MB improves performance; the default value is 4 MB. Use the command:

```
chatr +pi64M +pd64M  
/opt/WebSphere/AppServer/java/bin/PA_RISC2.0/native_threads/java
```

The command output provides the current operating system characteristics of the process executable.

For a number of HP-UX 11i kernel parameter recommendations see the InfoCenter article “Tuning operating systems”.

## Linux - Red Hat Advanced Server 2.1

Kernel updates for Red Hat Advanced Server 2.1 have implemented changes effecting WebSphere performance, especially memory-to-memory HTTP Session replication. If you are running any kernel prior to 2.4.9-e.23, upgrade at least to this kernel, but preferably to the latest supported.

## Linux - SUSE Linux Enterprise Server 8 SP2A

The Linux scheduler is very sensitive to excessive context switching, so fixes have been integrated into the SLES8 kernel distribution to introduce delay when a thread yields processing. This fix is automatically enabled in SLES8 SP3 but must be enable explicitly in SLES8 SP2A.

### ***sched\_yield\_scale tuning***

Enable this fix by running the `sysctl -w sched_yield_scale=1` command. If you are using a service pack below SP2A, upgrade to SP2A.

## Solaris

Tuning the following parameters has a significant performance impact for Solaris. They are set in the `startupServer.sh` script:

- ▶ Solaris TCP\_TIME\_WAIT\_INTERVAL
- ▶ Solaris TCP\_FIN\_WAIT\_2\_FLUSH\_INTERVAL
- ▶ Solaris TCP\_KEEPAIVE\_INTERVAL

When high connection rates occur, a large backlog of the TCP connections builds up and can slow server performance. The server can stall during certain peak periods. If this occurs, the `netstat` command will show that many of the sockets opened to port 80 were in the CLOSE\_WAIT or FIN\_WAIT\_2 state.

### ***Solaris file descriptors (ulimit)***

Specifies the number of open files permitted.

If the value of this parameter is too low, the error message Too many files open is displayed in the WebSphere Application Server `stderr.log`.

Check the UNIX reference pages on `ulimit` for the syntax for different shells. For KornShell (ksh) the command is:

```
ulimit -n 1024
```

Use the command `ulimit -a` to display the current values for all limitations on system resources.

The WebSphere Application Server `startupServer.sh` script sets this parameter to 1024 if its value is less than 1024.

### ***Solaris kernel semsys:seminfo\_semume***

The `semsys:seminfo_semume` kernel tuning parameter limits the Max Semaphore undo entries per process and needs to be greater than the default (10 on Solaris 7). Because this setting specifies a maximum value, the parameter does not cause any additional memory to be used unless it is needed.

This value is displayed as SEMUME if the `/usr/sbin/sysdef` command is run. There can be an entry in the `/etc/system` file for this tuning parameter. Set it via the `/etc/system` entry as follows:

```
set semsys:seminfo_semume = 1024
```

### ***Solaris kernel semsys:seminfo\_semopm***

This setting is displayed SEMOPM if the `/usr/sbin/sysdef` command is run. There can be an entry in the `/etc/system` file for this tuning parameter. Set it via the `/etc/system` entry as follows:

```
semsys:seminfo_semopm = 200
```

### ***Setting the virtual page size for WebSphere Application Server JVM***

On Solaris, there may be a desire to increase the virtual page size for the WebSphere Application Server JVM. The command is entered as follows:

```
chatr +pi64M +pd64M  
/opt/WebSphere/AppServer/java/bin/PA_RISC2.0/native_threads/java
```

where 64M stands for 64 MB, the recommended value.

### ***Other Solaris TCP parameters***

Customers have reported success with modifying other Solaris TCP parameters, including the following:

- ▶ `tcp_conn_req_max_q`
- ▶ `tcp_comm_hash_size`
- ▶ `tcp_xmit_hiwat`

Although significant performance differences have not been seen after raising these settings, the system might benefit.

Many other TCP parameters exist and can affect performance in a Solaris environment. For more information about tuning the TCP/IP Stack, see the article “Tuning your TCP/IP Stack and More” at:

<http://www.sean.de/Solaris/soltune.html>

### ***Windows NT or 2000 TCP/IP parameters***

Two important tuning parameters exist for Windows NT and Windows 2000:

- ▶ `TcpTimedWaitDelay`
- ▶ `MaxUserPort`

These parameters are set in the Windows registry.

**Important:** These two parameters should be used together when tuning WebSphere Application Server. Tuning one without tuning the other is not recommended.

### ***TcpTimedWaitDelay***

Determines the time that must elapse before TCP can release a closed connection and reuse its resources. This interval between closure and release is known as the TIME\_WAIT state or 2MSL (twice the maximum segment lifetime) state. During this time, reopening the connection to the client and server costs less than establishing a new connection. Reducing the value of this entry allows TCP to release closed connections faster, providing more resources for new connections.

Consider adjusting this value when the application that is running requires rapid release and creation of new connections, and there is a low throughput due to many connections sitting in TIME\_WAIT.

The default value is 0xF0 (240 seconds = 4 minutes), while the recommended value is the minimum value of 0x1E (30 seconds).

To view or set this parameter, use the **regedit** command, then access HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\TCP/IP\Parameters and create a new REG\_DWORD named TcpTimedWaitDelay. Set the value to decimal 30, which is Hex 0x0000001e. Then restart the system.

By using the **netstat** command, you will then be able to see that there are fewer connections in TIME\_WAIT.

### ***MaxUserPort***

Determines the highest port number TCP can assign when an application requests an available user port from the system. It is recommended that you set this value to at least decimal 32768.

Using the **regedit** command, access HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\TCP/IP\Parameters and create a new REG\_DWORD named MaxUserPort. Restart the system.

## **19.4.8 The Web server**

WebSphere Application Server provides plug-ins for several Web server brands and versions. Each Web server operating system combination has specific tuning parameters that affect the application performance.

This section discusses some of the performance tuning settings associated with the Web servers. In addition to the settings mentioned in this section, additional information about Web server tuning can be found in the WebSphere InfoCenter by expanding **Tuning -> Environment -> Web servers**. Then select **Tuning Web servers -> Web server tuning parameters**.

### **Web server configuration reload interval**

WebSphere Application Server administration tracks a variety of configuration information about WebSphere Application Server resources. Some of this information, such as URIs pointing to WebSphere Application Server resources, needs to be understood by the Web server. This configuration data is pushed to the Web server through the WebSphere Application Server plug-in at intervals specified by this parameter. Periodic updates allow new servlet definitions to be added without having to restart any of the WebSphere Application Server servers. However, the dynamic regeneration of this configuration information is costly in terms of performance. You can try to adjust this value in a stable production environment. The default reload interval setting is 60 seconds.

This parameter, `<RefreshInterval=xxxx>`, where `xxxx` is the number of seconds, is specified in the `<WAS_HOME>/config/plugin.xml` file. Increase the reload interval to a value that represents an acceptable wait time between the servlet update and the Web server update.

### **IBM HTTP Server 1.3**

The IBM HTTP Server v1.3.x (IHS) is a multi-process, single-threaded server (except on the Windows platform). Some factors related to the IBM HTTP Server performance have already been covered in “Web server” on page 861. This section provides additional information.

#### ***Modifying the WebSphere plug-in to improve performance***

You can improve the performance of IBM HTTP Server (with the WebSphere Web server plug-in) by modifying the plug-in's `RetryInterval` configuration. The `RetryInterval` is the length of time to wait before trying to connect to a server that has been marked temporarily unavailable. Making this change can help the IBM HTTP Server to scale higher than 400 users.

The plug-in marks a server temporarily unavailable if the connection to the server fails. Although the default value is 60 seconds, it is recommended that you lower this value in order to increase throughput under heavy load conditions. Lowering the `RetryInterval` is important for IBM HTTP Server 1.3 on UNIX operating systems that have a single thread per process, or for IBM HTTP Server 2.0 if it is configured to have fewer than 10 threads per process.



How can lowering the `RetryInterval` affect throughput? If the plug-in attempts to connect to a particular application server while the application server threads are busy handling other connections, which happens under heavy load conditions, the connection times out and the plug-in marks the server temporarily unavailable. If the same plug-in process has other connections open to the same server and a response is received on one of these connections, the server is marked again. However, when you use the IBM HTTP Server 1.3 on a UNIX operating system, there is no other connection since there is only one thread and one concurrent request per plug-in process. Therefore, the plug-in waits for the `RetryInterval` before attempting to connect to the server again.

Since the application server is not really down, but is busy, requests are typically completed in a small amount of time. The application server threads become available to accept more connections. A large `RetryInterval` causes application servers that are marked temporarily unavailable, resulting in more consistent application server CPU utilization and a higher sustained throughput.

**Note:** Although lowering the `RetryInterval` can improve performance, if all the application servers are running, a low value can have an adverse affect when one of the application servers is down. In this case, each IBM HTTP Server 1.3 process attempts to connect and fails more frequently, resulting in increased latency and decreased overall throughput.

### ***TCP initial connect timeout***

If, on the other hand, an application server is really unavailable because of a node outage, the time each plug-in process waits for a connect is defined by the underlying operating system's *TCP initial connect timeout*. So each plug-in process will have to wait for that timeout before marking an application server temporarily unavailable. This timeout value differs for every operating system. In Windows 2000 it is comparatively short and adjusted per TCP session to match the characteristics of the connection. For detailed information refer to Microsoft Knowledge Base Article #170359. On AIX this value can be queried and set using the `no` command. Its default value for AIX 5.2, for example, is 75 seconds.

In a scenario where a single-threaded, multi-process Web server is used (for example, IBM HTTP Server 1.3) the plug-in processes do not share information about application server status. Therefore each process will try to connect to the application server until it times out. Lowering the TCP initial connect timeout in AIX to a smaller value will improve this specific behavior. Keep in mind that, when changing this value, all processes using the TCP stack will be affected, and this might produce unwanted side effects. Using a value too small might accomplish the opposite, when connections are no longer established under high load because the client thinks his connection partner to be unavailable, when in fact the latter is just busy and would respond in due while. If you change this

parameter, be sure to perform a load test and watch how your systems behave under peak load! For detailed instructions on using **no**, refer to the AIX man-page or the administration manual.

**Note:** The value for `tcp_keepinit` has to be specified in *halfseconds*. Refer to Example 19-5, where the TCP initial connect timeout value is set from 75 to 10 seconds. For instructions on how to accomplish this in other operating systems, please refer to their administration manuals, respectively.

---

*Example 19-5 Querying and setting network options in AIX*

---

```
$ no -o tcp_keepinit
tcp_keepinit = 150

$ no -o tcp_keepinit=20

$ no -o tcp_keepinit
tcp_keepinit = 20
```

---

**Attention:** Since WebSphere Application Server V5.x the Web server plug-in supports the new `ConnectTimeout` parameter. This attribute is valid inside the `<server>` tag of `plugin-cfg.xml` and lets you bypass the operating system's TCP initial connect timeout. You can specify how long to wait until the currently queried application server is marked unavailable.

This is the recommended procedure and preferred to changing the operating system's TCP initial connect timeout value. Refer to 5.7.3, "Tuning failover" on page 207 for a detailed description of the `ConnectTimeout` parameter.

### ***Access logs***

All incoming HTTP requests are logged here. Logging degrades performance because I/O operation overhead causes logs to grow significantly in a short time.

To turn logging on or off, edit the IBM HTTP Server `httpd.conf` file, located in the directory `<IBM HTTP Server Home>/conf`. Search for a line with the text `CustomLog`. Comment out this line, then save and close the `httpd.conf` file. Stop and restart the IBM HTTP Server. By default, logging is enabled, but for better performance it is recommended that you disable the access logs.

### ***MaxClients/ThreadsPerChild***

The value of the `MaxClients` parameter can significantly impact the application, particularly if it is too high. More is not always better. The optimum value depends on the application. Setting `MaxClients` too high can cause swapping if

there is insufficient RAM. Set this value to prevent bottlenecks, allowing just enough traffic through to the application server.

The maximum number of concurrent requests accepted by the IBM HTTP Server is configured by a parameter in the `httpd.conf` configuration file. The parameter sets the number of concurrent threads running at any one time within the IBM HTTP Server. The default value is 150.

The parameter keyword is different on different platforms. On IBM HTTP servers running on the UNIX platform, the keyword is `MaxClients`. On the Windows platform, the configuration parameter keyword is `ThreadsPerChild`. Refer to “ThreadsPerChild” on page 892 for more information about this parameter on Windows platforms.

To configure the number of maximum concurrent allowed connections, open the `httpd.conf` file and change the value of `ThreadsPerChild` (Windows) or `MaxClients` (UNIX) to the appropriate value, as seen in Example 19-6. Save the changes and restart the IBM HTTP server.

---

*Example 19-6 ThreadsPerChild/MaxClients parameter in httpd.conf*

---

```
# Number of concurrent threads at a time (set the value to more or less
# depending on the responsiveness you want and the resources you wish
# this server to consume).
```

```
MaxClients 50
```

---

### ***MinSpareServers, MaxSpareServers, and StartServers***

Pre-allocates and maintains the specified number of processes so that few processes are created and destroyed as the load approaches the specified number of processes (based on `MinSpareServers`). Specifying similar values reduces the CPU usage for creating and destroying HTTPD processes. Adjust this parameter if the time waiting for IBM HTTP Server to start more servers so that it can handle HTTP requests is not acceptable.

For optimum performance, specify the same value for the `MaxSpareServers` and the `StartServers` parameters. If `MaxSpareServers` is set to less than `MinSpareServers`, IBM HTTP Server resets `MaxSpareServer=MinSpareServer+1`. Setting the `StartServers` too high can cause swapping if memory is not sufficient, degrading performance.

To view or change these values, edit the following directives in the file `httpd.conf`, located in the directory `<IBM HTTP Server Home>/conf`:

- ▶ `MinSpareServers`
- ▶ `MaxSpareServers`
- ▶ `StartServers`

Default values are MinSpareServers 5, MaxSpareServers 10, and StartServers 5.

For more information about tuning IHS, see “Hints on Running a High-Performance Web Server” at:

<http://www.ibm.com/software/webserver/httpserver/doc/v136/misc/perf.html>

## IBM HTTP Server 1.3 - Linux

The following instructions are important when running the IBM HTTP Server on Linux.

### *MaxRequestsPerChild*

The MaxRequestsPerChild directive sets the limit on the number of requests that an individual child server process handles. After the number of requests reaches the value set for the MaxRequestsPerChild parameter, the child process dies. If there are no known memory leaks with Apache and Apache's libraries, set this value to zero (0).

To change this value, edit the IBM HTTP server file httpd.conf located in the directory <IBM HTTP Server Home>/conf. Change the value of the MaxRequestsPerChild parameter. Save the changes and restart the IBM HTTP server. By default, this value is set to 500.

## IBM HTTP Server 1.3 - Windows 2000 or Windows 2003

This section discusses the IBM HTTP Server tuning specifics when running on Windows NT or 2000. In contrary to other platforms, IHS is a single-process, multi-threaded server on the Windows platform.

### *ThreadsPerChild*

This parameter sets the number of concurrent threads running at any one time within the IBM HTTP Server. See “MaxClients/ThreadsPerChild” on page 890 for information about how to change this value.

There are two ways to find how many threads are in use. They are as follows:

1. Use the Windows 2000 or 2003 Task Manager:
  - a. Right-click the **Status bar** on your desktop. Select **Task Manager**.
  - b. Select the **Processes** tab.
  - c. Click **View -> Select Columns**.
  - d. Select **Thread Count**.
  - e. Click **OK**.

2. Use IBM HTTP Server server-status (this choice works on all platforms, not just Windows 2000 or Windows 2003.). To enable it, do the following depending on your Web server version:
  - a. IBM HTTP Server 1.3.x:
    - i. Edit httpd.conf located in the directory <IBM\_HTTP\_Server\_Home>/conf and remove the comment character "#" from the lines shown in Example 19-7.

*Example 19-7 Enabling HTTP Server 1.3 server-status in httpd.conf file*

---

```
#LoadModule status_module modules/ApacheModuleStatus.dll
#<Location/server-status>
#SetHandler server-status
#</Location>
```

---

- ii. Save the changes and restart the IBM HTTP server.
- iii. In a Web browser, go to the URL  
`http://<your_host>/server-status`  
and click **Reload to update status**.
- iv. Alternatively, if the browser supports refresh, go to  
`http://<your_host>/server-status?refresh=5`  
to refresh every 5 seconds.
- b. IBM HTTP Server 2.0:
  - i. Edit httpd.conf located in the directory <IBM\_HTTP\_Server\_Home>/conf and remove the comment character "#" from the lines shown in Example 19-8.

*Example 19-8 Enabling HTTP Server 2.0 server-status in httpd.conf file*

---

```
#LoadModule status_module modules/mod_status.so#
#<Location /server-status>
# SetHandler server-status
#</Location>
```

---

- ii. Save the changes and restart the IBM HTTP server.
- iii. In a Web browser, go to the URL  
`http://<your_host>/server-status`  
and click **Reload to update status**.
- iv. Alternatively, if the browser supports refresh, go to  
`http://<your_host>/server-status?refresh=5`  
to refresh every 5 seconds.

The default value of the `ThreadsPerChild` parameter is 50 for IBM HTTP Server 1.3.28, and 250 for IBM HTTP Server 2.0.

**Note:** If using the server-status module on other platforms, additional changes may be needed, and modules will not be in the form of a DLL. Consult the administrator's guide for more specific details.

## IBM HTTP Server 2.0

Although IHS 1.3.28 is installed as the default Web server if you select a full installation of WebSphere Application Server V5.1, IBM introduced plug-in support for IHS 2.0 in Application Server V5.0. If you want to use the IHS 2.0 Web server, you first have to download and install IHS 2.0 from

<http://www.ibm.com/software/webservers/httpservers/>

For details on installing and upgrading from IHS V1.3.x to V2.0.x, refer to the following InfoCenter article: navigate to **Installation -> Getting Started -> Preparing to install and configure a Web server**. There, select the topic **Installing IBM HTTP Server powered by Apache 2.0**. The easiest and recommended way is to install IHS 2.0 before installing WebSphere Application Server. Then select the plug-in for version 2.0 to have the installer configure IHS 2.0 to work with WebSphere Application Server V5.1.

First we give you a short introduction into IBM HTTP Server 2.0, for example on the new features. For IHS 2.0 tuning information go to “Tuning IHS 2.0” on page 897.

### New Features

There are various changes in IHS 2.0, and this section describes specific issues to consider to help you determine whether to migrate your system from IHS 1.3 to 2.0. Following are a few important new features of IHS 2.0. For a complete feature list, visit the Apache 2.0 Web site at

[http://httpd.apache.org/docs-2.0/new\\_features\\_2\\_0.html](http://httpd.apache.org/docs-2.0/new_features_2_0.html)

#### ► IHS 2.0 is a thread-based Web server

Although IHS 1.3 is thread-based on the Windows platform, it is a process-based Web server on all Unix and Linux platforms. That means, it implements the *multi-process, single-thread* process model: for each incoming request, a new child process is created or requested from a pool to handle it.

However, IHS 2.0 is now a fully thread-based Web server on all platforms. This gives you the following advantages:

- Each request does not require its *own* HTTPD process anymore, and less memory is needed.

- Overall performance improves because in most cases new HTTPD processes do not need to be created.
- The plug-in load-balancing and failover algorithms work more efficiently in a single-process, multi threaded HTTP server. If one plug-in thread marks an application server unavailable, all other connection threads of the plug-in will share that knowledge, and will not try to connect to this particular application server again before the `RetryInterval` has elapsed. See 5.7, “Web server plug-in behavior and failover” on page 184 for information on the `RetryInterval` parameter and plug-in load balancing and failover issues.

**Note:** On the Unix platform, IHS 2.0 also allows you to configure more than one process to be started.

► The *mod\_deflate* module

This module allows supporting browsers to request that content be compressed before delivery. It provides the *Deflate* output filter that lets output from the server be compressed before it is sent to the client over the network. Some of the most important benefits of using the *mod\_deflate* module are:

- Saves network bandwidth during data transmission
- Shortens data transmission time
- Generally improves overall performance

Detailed information about configuring and using *mod\_deflate* can be found at

[http://httpd.apache.org/docs-2.0/mod/mod\\_deflate.html](http://httpd.apache.org/docs-2.0/mod/mod_deflate.html)

► Request and response filtering

Apache (and IHS) modules may now be written as filters which act on the stream of content as it is delivered to or from the server.

► No GUI-based administration tool

IHS 2.0 does not contain a GUI-based administration tool anymore. Changes have to be performed manually on the `httpd.conf` file using an editor. If browser based administration is imperative for you, you should not upgrade to IHS 2.0.

### **Configuration migration**

Be sure to review your IHS 1.3.x `httpd.conf` configuration file to see what modules are loaded and which directives are set, because of the following reasons:

- Many IHS 1.3 directives remain unchanged in IHS 2.0. Notable exceptions relate primarily to new IHS 2.0 functions, such as filtering and Unix thread support.

- ▶ IHS 2.0 contains some new directives.
- ▶ In IHS 2.0, some 1.3 directives have been deprecated.

**Important:** Do not overwrite the new IHS 2.0 httpd.conf with your IHS 1.3.x one because this will not work.

After installing WebSphere Application Server V5.1, verify that all needed WebSphere plug-in directives were added to the IHS 2.0 configuration file.

The following examples show the WebSphere plug-in directives that must be added to httpd.conf for IHS 2.0 running on Unix (see Example 19-9; on the Windows platform this is similar).

*Example 19-9 Plug-in directives that must be added to httpd.conf for IHS 2.0 on Unix.*

---

```
Alias /IBMWebAS/ <wasroot>/web/  
Alias /WSsamples <wasroot>/WSsamples/  
LoadModule was_ap20_module <wasroot>/bin/mod_was_ap20_http.so  
WebSpherePluginConfig <wasroot>/config/cells/plugin-cfg.xml
```

---

### ***Single-processing versus multi-processing***

Apache version 2 (and thus IHS 2) achieves efficient support of different operating systems by implementing a Multi-Processing Modules (MPM) architecture, allowing it, for example, to use native networking features instead of going through an emulation layer in version 1.3. For detailed information about MPM refer to the Apache HTTP Server documentation found at:

<http://httpd.apache.org/docs-2.0/mpm.html>

MPMs are chosen at compile time and differ from operating system, which implies that the Windows version uses a different MPM module than the AIX or Linux version. The default MPM for Windows is `mpm_winnt`, whereas the default module for AIX is `mpm_worker`. For a complete list of available MPMs, refer to the Apache MPM documentation URL above. To identify which was MPM compiled into an Apache 2.0 or IHS 2.0 Web server, run the **httpd -l** command, which prints out the module names. Look for a module name `worker`, or a name starting with the `mpm` prefix (see Example 19-10).

*Example 19-10 Listing of compiled in modules for IHS 2.0 on AIX*

---

```
root@app2:/usr/IBMIHS/bin $ ./httpd -l  
  
Compiled in modules:  
  core.c  
  worker.c  
  http_core.c
```



mod\_suexec.c  
mod\_so.c

---

► **mpm\_winnt module**

This Multi-Processing Module is the default for the Windows operating systems. It uses a single control process which launches a single child process which in turn creates all the threads to handle requests.

► **mpm\_worker module**

This Multi-Processing Module implements a hybrid multi-process multi-threaded server. This is the default module for AIX. By using threads to serve requests, it is able to serve a large number of requests with less system resources than a process-based server. Yet it retains much of the stability of a process-based server by keeping multiple processes available, each with many threads.

### ***Tuning IHS 2.0***

This section gives you configuration tips for the Unix and Windows platform that provide a good starting point for Web server tuning, and an explanation of the new configuration directives used. Keep in mind that every system and every site has different requirements, so make sure to adapt these settings to your needs! See Example 19-11 on page 898 for a Unix sample configuration and Example 19-12 on page 898 for a Windows sample configuration.

► **ThreadsPerChild**

Each child process creates a fixed number of threads as specified in the `ThreadsPerChild` directive. The child creates these threads at startup and never creates more. If using an MPM like `mpm_winnt`, where there is only one child process, this number should be high enough to handle the entire load of the server. If using an MPM like `worker`, where there are multiple child processes, the total number of threads should be high enough to handle the common load on the server.

► **ThreadLimit**

This directive sets the maximum configured value for `ThreadsPerChild` for the lifetime of the Apache process. `ThreadsPerChild` can be modified during a restart up to the value of this directive.

► **MaxRequestsPerChild**

This directive controls after how many requests a child server process is recycled and a new one is launched.

► **MaxClients**

This controls the maximum total number of threads that may be launched.

- ▶ **StartServers**  
The number of processes that will initially be launched is set by the StartServers directive.
- ▶ **MinSpareThreads and MaxSpareThreads**  
During operation, the total number of idle threads in all processes will be monitored, and kept within the boundaries specified by MinSpareThreads and MaxSpareThreads.
- ▶ **ServerLimit**  
The maximum number of processes that can be launched is set by the ServerLimit directive.

*Example 19-11 A sample configuration for the Unix platform*

---

```
<IfModule worker.c>
ServerLimit 1
ThreadLimit 2048
StartServers 1
MaxClients 1024
MinSpareThreads 1
MaxSpareThreads 1024
ThreadsPerChild 1024
MaxRequestsPerChild 0
</IfModule>
```

---

**Attention:** Using a ThreadsPerChild value greater than 512 is not recommended on the Linux and Solaris platform. If 1024 threads are needed, the recommended solution is to increase the ServerLimit value to 2 to launch two server process with 512 threads each.

*Example 19-12 A sample configuration for the Windows platform*

---

```
<IfModule mpm_winnt.c>
ThreadsPerChild 2048
MaxRequestsPerChild 0
</IfModule>
```

---

## **Sun ONE Web Server (formerly iPlanet)**

The default configuration of the Sun ONE Web server, Enterprise Edition provides a single-process, multi-threaded server.

### **Active Threads**

This value specifies the current number of threads active in the server. After the server reaches the limit set with this parameter, the server stops servicing new

connections until it finishes old connections. Thus, if this setting is too low, the server can become throttled, resulting in degraded response times.

To tell if the Web server is being throttled, consult its perfdump statistics. Look at the following data:

- ▶ **WaitingThreads** count: If the **WaitingThreads** count is getting close to zero, or is zero, the server is not accepting new connections.
- ▶ **BusyThreads** count: If the **WaitingThreads** count is close to zero, or is zero, **BusyThreads** is probably very close to its limit.
- ▶ **ActiveThreads** count: If the **ActiveThreads** count is close to its limit, the server is probably limiting itself.

Use the Maximum number of simultaneous requests parameter in the Enterprise Server Manager interface to control the number of active threads within Sun ONE Web server, Enterprise Edition. This setting corresponds to the **RqThrottle** parameter in the **magnus.conf** file.

The default value is 512. It is recommended that you increase the thread count until the active threads parameters show optimum behavior.

## Microsoft IIS

The Web server has several properties that dramatically affect the performance of the application server.

### ***IIS permission properties***

The default settings are usually acceptable. However, because other products can change the default settings without the user's knowledge, make sure that you check the IIS settings for the Home Directory permissions of the Web server. The permissions should be set to **Script** and not to **Execute**. If the permissions are set to **Execute**, no error messages are returned, but the performance of WebSphere Application Server is decreased.

To check or change these permissions, perform the following procedure in the Microsoft management console:

1. Select the Web site (usually the default Web site). Right-click and select the **Properties** option. Open the **Home Directory** tab.
2. Go to the **Application** settings and ensure that the **Script** check box is selected in the Permissions list and that the **Execute** check box is cleared.

**Note:** It may be necessary to check the permissions of the sePlugin to confirm that the Execute permissions are set to Execute. This is done from the same console by expanding the Web server and selecting the properties for the sePlugin object.

### ***Number of expected hits per day***

This parameter controls the memory that IIS allocates for connections.

Using the performance window, set the parameter to “More than 100000” in the Web site properties panel of the Microsoft management console. The default for this value is “Fewer than 100000”.

### ***ListenBackLog parameter***

If using IIS on Windows 2000 or Windows 2003, the server is likely to encounter failed connections under heavy load conditions (typically more than 100+ clients). This condition commonly results from IIS rejecting connections. Alleviate the condition by using the ListenBackLog parameter to increase the number of requests IIS keeps in its queue. If you intermittently receive the error message “Unable to locate server” in a Netscape browser when running a heavy load, consider raising this parameter.

Use the **regedit** command to access the registry. In the registry window, locate the parameter in the HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\ListenBackLog directory. Right-click the parameter to modify it. Adjust the setting according to the server load.

The default value is 25 (decimal). The ListenBackLog parameter can be set as high as 200 without negative impact on performance and an improvement in load handling.

## **19.4.9 Dynamic Cache Service**

The Dynamic Cache Service improves performance by caching the output of servlets, commands and JavaServer Pages (JSP) files. WebSphere Application Server consolidates several caching activities, including servlets, Web services, and WebSphere commands into one service called the dynamic cache. These caching activities work together to improve application performance, and share many configuration parameters, which are set in an application server's dynamic cache service.

The dynamic cache works within an application server Java Virtual Machine (JVM), intercepting calls to cacheable objects, for example through a servlet's service() method or a command's execute() method, and either stores the

object's output to or serves the object's content from the dynamic cache. Because J2EE applications have high read/write ratios and can tolerate small degrees of latency in the currency of their data, the dynamic cache can create an opportunity for significant gains in server response time, throughput, and scalability.

See 14.2, "Using WebSphere dynamic cache services" on page 534 for an in-depth discussion on dynamic caching, and 16.7, "Dynamic Cache Monitor" on page 729 on using the dynamic cache monitor.

## 19.4.10 Security settings

This section discusses how various settings related to security affect performance. Refer to *IBM WebSphere V5.0 Security*, SG24-6573 for more information about WebSphere security.

### ***Disabling security***

Security is a global setting. When security is enabled, performance can be decreased between 10-20%. Therefore, disable security when not needed.

In the Administrative Console, select **Security -> Global Security**. The Enabled and Enforce Java 2 Security check boxes control global security settings. By default, security is not enabled.

### ***Fine-tune the security cache timeout for the environment***

If WebSphere Application Server security is enabled, the security cache timeout can influence performance. The timeout parameter specifies how often to refresh the security-related caches.

Security information pertaining to beans, permissions, and credentials is cached. When the cache timeout expires, all cached information becomes invalid. Subsequent requests for the information result in a database lookup. Sometimes, acquiring the information requires invoking a Lightweight Directory Access Protocol (LDAP)-bind or native authentication. Both invocations are relatively costly operations for performance.

Determine the best trade-off for the application by looking at usage patterns and security needs for the site.

Use the Administrative Console to change this value. To do so, select **Security -> Global Security**. Enter an appropriate value in seconds in the Cache Timeout field. The default is 600 seconds.

## 19.4.11 Tuning Secure Sockets Layer

The following are two types of Secure Sockets Layer (SSL) performance:

- ▶ Handshake
- ▶ Bulk encryption/decryption

### Overview of handshake and bulk encryption and decryption

When an SSL connection is established, an SSL handshake occurs. After a connection is made, SSL performs bulk encryption and decryption for each read/write. The performance cost of an SSL handshake is much larger than that of bulk encryption and decryption.

### How to enhance SSL performance

In order to enhance SSL performance, the number of individual SSL connections and handshakes must be decreased.

Decreasing the number of connections increases performance for secure communication through SSL connections, as well as non-secure communication through simple TCP connections. One way to decrease individual SSL connections is to use a browser that supports HTTP 1.1. Decreasing individual SSL connections could be impossible for some users if they cannot upgrade to HTTP 1.1.

Another common approach is to decrease the number of connections (both TCP and SSL) between two WebSphere Application Server components. The following guidelines help to ensure the HTTP transport of the application server is configured so that the Web server plug-in does not repeatedly reopen new connections to the application server:

- ▶ The maximum number of keep-alives should be, at minimum, as large as the maximum number of requests per thread of the Web server (or maximum number of processes for IHS on UNIX). In other words, make sure the Web server plug-in is capable of obtaining a keep-alive connection for every possible concurrent connection to the application server. Otherwise, the application server will close the connection after a single request has been processed. This adds performance overhead due to the creation and destruction of the connection, and the handshaking associated with SSL transactions. Also, the maximum number of threads in the Web container thread pool should be larger than the maximum number of keep-alives, in order to prevent the Web container threads from being consumed with keep-alive connections.
- ▶ The maximum number of requests per keep-alive connection can also be increased. The default value is 100, which means the application server will close the connection from the plug-in after 100 requests. The plug-in would

then have to open a new connection. The purpose of this parameter is to prevent denial of service attacks when connecting to the application server and continuously sending requests in order to tie up threads in the application server.

- Use a hardware accelerator if the system performs several SSL handshakes.

Hardware accelerators currently supported by WebSphere Application Server only increase the SSL handshake performance, not the bulk encryption/decryption. An accelerator typically only benefits the Web server because Web server connections are short-lived. All other SSL connections in WebSphere Application Server are long-lived.

- Use an alternative cipher suite with better performance.

The performance of a cipher suite is different with software and hardware. Just because a cipher suite performs better in software does not mean a cipher suite will perform better with hardware. Some algorithms are typically inefficient in hardware (for example, DES and 3DES). However, specialized hardware can provide efficient implementations of these same algorithms.

The performance of bulk encryption and decryption is affected by the cipher suite used for an individual SSL connection.

## 19.4.12 Object Request Broker (ORB)

Several settings are available for controlling internal Object Request Broker (ORB) processing. You can use these to improve application performance in the case of applications containing enterprise beans.

You can change these settings for the default server or any application server configured in the administrative domain from the Administrative Console.

### ***Pass by value versus Pass by reference (com.ibm.CORBA.iiop.noLocalCopies)***

For EJB 1.1 beans, the EJB 1.1 specification states that method calls are to be Pass by value. For every remote method call, the parameters are copied onto the stack before the call is made. This can be expensive. The Pass by reference, which passes the original object reference without making a copy of the object, can be specified.

For EJB 2.0 beans, interfaces can be local or remote. For local interfaces, method calls are Pass by reference, by default.

If the EJB client and EJB server are installed in the same WebSphere Application Server instance, and the client and server use remote interfaces, specifying Pass by reference can improve performance up to 50%.

Please note that Pass by reference helps performance only when non-primitive object types are being passed as parameters. Therefore, int and floats are always copied, regardless of the call model.

**Important:** Pass by reference can be dangerous and can lead to unexpected results. If an object reference is modified by the remote method, the change might be seen by the caller.

Use the Administrative Console to set this value:

3. Select **Servers -> Application Servers**.
4. Select the application server you wish to change.
5. Then, select **ORB Service** from Additional Properties.
6. Select the check box **Pass by Reference**.
7. Click **OK** and **Apply** to save the changes.
8. Stop and restart the application server.

The default is Pass by value for remote interfaces and Pass by reference for EJB 2.0 local interfaces.

If the application server expects a large workload for enterprise bean requests, the ORB configuration is critical. Take note of the following properties.

#### ***com.ibm.CORBA.ServerSocketQueueDepth***

This property corresponds to the length of the TCP/IP stack listen queue and prevents WebSphere Application Server from rejecting requests when there is no space in the listen queue.

If there are many simultaneous clients connecting to the server-side ORB, this parameter can be increased to support the heavy load up to 1000 clients. The default value is 50.

To set the property (in our example we set it to 200), follow these steps:

1. Select **Servers -> Application Servers**.
2. Click the application server you want to tune.
3. Select **Process Definition** under Additional Properties.
4. Select **Java Virtual Machine** under Additional Properties.
5. Enter `-Dcom.ibm.CORBA.ServerSocketQueueDepth=200` in the Generic JVM Properties field.



### ***Object Request Broker connection cache maximum (com.ibm.CORBA.MaxOpenConnections)***

This property has two names and corresponds to the size of the ORB connection table. The property sets the standard for the number of simultaneous ORB connections that can be processed.

If there are many simultaneous clients connecting to the server-side ORB, this parameter can be increased to support the heavy load up to 1000 clients. The default value is 240. To change this value:

1. Select **Servers -> Application Servers**.
2. Select the application server you want to tune.
3. Select **ORB Service** under Additional Properties.
4. Update the Connection cache maximum field and click **OK**.
5. Click **Apply** to save the changes then restart the application server.

### ***ORB thread pool size***

Refer to “EJB container” on page 854 for more information.

## **19.4.13 XML parser selection**

Add XML parser definitions to the `jaxp.properties` file and `xerces.properties` file found in the `<WAS_HOME>/jre/lib` directory to help facilitate server startup. The `XMLParserConfiguration` value might have to be changed as new versions of Xerces are provided.

In both files, insert the lines shown in Example 19-13.

#### *Example 19-13 XML parser definitions*

---

```
javax.xml.parsers.SAXParserFactory=org.apache.xerces.jaxp.SAXParserFactoryImpl
javax.xml.parsers.DocumentBuilderFactory=org.apache.xerces.jaxp.DocumentBuilderFactoryImpl
org.apache.xerces.xni.parser.XMLParserConfiguration=org.apache.xerces.parsers.
StandardParserConfiguration
```

---

## **19.4.14 Transaction service settings - transaction log**

When an application running on WebSphere accesses more than one resource, WebSphere stores transaction information to properly coordinate and manage the distributed transaction. In a higher transaction load, this persistence slows down performance of the application server due to its dependency on the operating system and the underlying storage systems.

To achieve better performance, move the transaction log files to a storage device with more physical disk drives, or preferably RAID disk drives. When the log files are moved to the file systems on the RAID disks, the task of writing data to the physical media is shared across the multiple disk drives. This allows more concurrent access to persist transaction information and faster access to that data from the logs. Depending upon the design of the application and storage subsystem, performance gains can range from 10% to 100%, or even more in some cases.

This change is applicable only to the configuration where the application uses distributed resources or XA transactions, for example multiple databases and resources are accessed within a single transaction. Consider setting this property when the application server shows one or more of the following signs:

- ▶ CPU utilization remains low despite an increase in transactions.
- ▶ Transactions fail with several timeouts.
- ▶ Transaction rollbacks occur with “unable to enlist transaction” exception.
- ▶ Application server hangs in middle of a run and requires the server to be restarted.
- ▶ The disk on which an application server is running shows higher utilization.

Use the WebSphere Administrative Console to change the location of the transaction logs. Select **Servers -> Application Servers -> <AppServer name>**. Then select **Transaction Service** from the Additional Properties pane. Enter the new path in the Transaction log directory field. By default, the transaction log has a size of 1 MB and is stored in the directory <WAS\_HOME>/tranlog/<AppServer name>.

It is recommended that you create a file system with at least three to four disk drives RAIDed together in a RAID-0 configuration. Then, create the transaction log on this file system with the default size. When the server is running under load, check the disk input and output. If disk input and output time is more than 5%, consider adding more physical disks to lower the value. If disk input and output is low, but the server is still high, consider increasing the size of the log files.

### 19.4.15 DB2 tuning

DB2 has many parameters that can be configured to optimize database performance. For complete DB2 tuning information, refer to the *DB2 UDB Administration Guide: Performance*.

## **DB2 logging**

DB2 has corresponding log files for each database. Performance improvements can be gained by setting the log files on a different hard drive from the database files. Refer to “AIX with DB2” on page 883 for more information.

## **DB2 configuration advisor**

Located in the DB2 Control Center, this advisor calculates and displays recommended values for the DB2 buffer pool size, the database and database manager configuration parameters, with the option of applying these values. See more information about the advisor in the online help facility within the Control Center.

## **Use TCP sockets for DB2 on Linux**

On Linux platforms, whether the DB2 server resides on a local machine with WebSphere Application Server or on a remote machine, configure the DB2 application databases to use TCP sockets for communications with the database.

The directions for configuring DB2 on Linux can be found in the WebSphere Application Server installation documentation for the various operating systems. This document specifies setting DB2COMM for TCP/IP and corresponding changes required in the /etc/services file.

The default is to use shared memory for local databases but it is recommended that you change the specification for the DB2 application databases and for any session databases from shared memory to TCP sockets.

## **DB2 MaxAppls and DB2 MaxAgents**

When configuring the data source settings for the databases, confirm the DB2 MaxAppls setting is greater than the maximum number of connections for the data source. If you are planning to use multiple cluster members, set the MaxAppls value as the maximum number of connections multiplied by the number of cluster members.

The same relationship applies to the session manager number of connections. The MaxAppls setting must be equal to or greater than the number of connections. If you are using the same database for session and data sources, set the MaxAppls value as the sum of the number of connection settings for the session manager and the data sources.

Refer to “Connection pool size” on page 852 for more information about minimum and maximum number of connections.

## DB2 buffpage

Buffpage is a database configuration parameter. A buffer pool is a memory storage area where database pages containing table rows or index entries are temporarily read and changed. The purpose of the buffer pool is to improve database system performance. Data can be accessed much faster from memory than from disk.

Refer to the WebSphere InfoCenter section “DB2 tuning parameters” for more information about how to configure this parameter.

## DB2 query optimization level

When a database query is executed in DB2, various methods are used to calculate the most efficient access plan. The query optimization level parameter sets the amount of work and resources that DB2 puts into optimizing the access plan. The range is from zero to 9.

An optimization level of 9 causes DB2 to devote a lot of time and all of its available statistics to optimizing the access plan.

The optimization level is set on individual databases and can be set with either the command line or with the DB2 Control Center. Static SQL statements use the optimization level specified on the **prep** and **bind** commands. If the optimization level is not specified, DB2 uses the default optimization as specified by the `dft_queryopt` parameter. Dynamic SQL statements use the optimization class specified by the current query optimization special register, which is set using the SQL Set statement. For example, the following statement sets the optimization class to 1:

```
Set current query optimization = 1
```

If the current query optimization register has not been set, dynamic statements will be bound using the default query optimization class.

The default value is 5. It is recommended that you set the optimization level for the needs of the application. High levels should only be used when there are very complicated queries.

## DB2 reorgchk

The performance of the SQL statements can be impaired after many updates, deletes, or inserts have been made. Performance can be improved by obtaining the current statistics for the data and rebinding.

Use the following DB2 command to issue runstats on all user and system tables for the database you are currently connected to:

```
reorgchk update statistics on table all
```

You should then rebind packages using the **bind** command.

In order to see if runstats has been done, issue the following command on DB2 CLP:

```
db2 -v "select tname, nleaf, nlevels, stats_time from sysibm.sysindexes"
```

If no runstats has been done, nleaf and nlevels will be filled with -1 and stats\_time will have an empty entry "-". If runstats was done already, the real-time stamp when the runstats was completed will also be displayed under stats\_time. If you think the time shown for the previous runstats is too old, execute runstats again.

## DB2 MinCommit

This parameter allows delayed writing of log records to a disk until a minimum number of commits have been performed, reducing the database manager overhead associated with writing log records. For example, if MinCommit is set to 2, a second commit would cause output to the transaction log for the first and second commits. The exception occurs when a one-second timeout forces the first commit to be output if a second commit does not come along within one second. In test applications, up to 90% of the disk input and output was related to the DB2 transaction log. Changing MinCommit from 1 to 2 reduced the results to 45%.

Try to adjust this parameter if the disk input/output wait is more than 5% and there is DB2 transaction log activity from multiple sources. When a lot of activity occurs from multiple sources, it is less likely that a single commit will have to wait for another commit (or the one-second timeout).

*Do not* adjust this parameter if you have an application with a single thread performing a series of commits (each commit could hit the one-second delay).

To view the current value for a particular database follow these steps:

- ▶ Issue the DB2 command **get db cfg for <dbname>** (where <dbname> is the name of the application database) to list database configuration parameters.
- ▶ Look for "Group commit count (MINCOMMIT)".
- ▶ Set a new value by issuing the DB2 command **update db cfg for <dbname> using mincommit n** (where n is a value between 1 and 25 inclusive).

The new setting takes effect immediately.

The following are several metrics that are related to DB2 MinCommit:

- ▶ The disk input/output wait can be observed on AIX with the command **vmstat 5**. This shows statistics every 5 seconds. Look for the *wa* column under the CPU area.
- ▶ The percentage of time a disk is active can be observed on AIX with the command **iostat 5**. This shows statistics every 5 seconds. Look for the *%tm\_act* column.
- ▶ The DB2 command **get snapshot for db on <dbname>** (where <dbname> is the name of the application database) shows counters for log pages read and log pages written.

The default value is 1. It is recommended that you set MinCommit to 1 or 2 (if the circumstance permits).

## 19.4.16 Additional reference materials

- ▶ IBM WebSphere Application Server Library, including monitoring and troubleshooting documentation, are found at:  
<http://www.ibm.com/software/webservers/appserv/library/index.html>
- ▶ WebSphere Application Server White papers found at:  
<http://www.ibm.com/support/search.wss?rs=180&tc=SSEQTP&dc=DB100>  
<http://www-306.ibm.com/software/webservers/appserv/whitepapers.html>
- ▶ iSeries performance documents, including *WebSphere Application Server for iSeries Performance Considerations* and links to the PTDV tool, Workload Estimator tool, and other documents are found at:  
<http://www.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/product/PerformanceConsiderations.html>
- ▶ *IBM WebSphere Application Server Advanced Edition Tuning Guide* (Version 4.02) found at:  
[http://www.ibm.com/software/webservers/appserv/doc/v40/ae/infocenter/was/pdf/nav\\_Tuneguide.pdf](http://www.ibm.com/software/webservers/appserv/doc/v40/ae/infocenter/was/pdf/nav_Tuneguide.pdf)
- ▶ “J2EE Application Development: One or many applications per application server?” (Nov. 2002) found at:  
[http://www7b.software.ibm.com/wsdd/techjournal/0211\\_alcott/alcott.html](http://www7b.software.ibm.com/wsdd/techjournal/0211_alcott/alcott.html)
- ▶ “Handling Static content in WebSphere Application Server” (Nov. 2002) found at:  
[http://www7b.software.ibm.com/wsdd/techjournal/0211\\_brown/brown.html](http://www7b.software.ibm.com/wsdd/techjournal/0211_brown/brown.html)
- ▶ WebSphere Best Practices Web site found at:  
<http://www7b.software.ibm.com/wsdd/zones/bp/>

- The WebSphere Application Server zone at WebSphere Developer Domain at:

<http://www7b.boulder.ibm.com/wsdd/zones/was/>







## Part 6

# Appendixes





## Backup/recovery of Network Deployment configuration

This appendix describes two simple and tested procedures to backup and restore the configuration data of a Network Deployment configuration in the case of node failure caused for example by a harddisk corruption or similar. It explains the steps required to restore a node to a previously saved configuration state using either a filesystem backup or the command-line tools **backupConfig** and **restoreConfig**.

**Note:** These procedures should not to be seen as a replacement for established and more powerful (and more expensive) backup scenarios for the entire system using backup tools like IBM Tivoli Storage Manager or other third party backup mechanisms. Only WebSphere Application Server configuration data will be restored. Other data typically required for a production cell, such as the SSL keyring, is not backed up or restored by the WebSphere supplied utilities.

## Network Deployment configurations

In a typical Network Deployment cluster configuration, capacity and reliability can be increased by horizontal scaling, for example the usage of multiple physical nodes inside a cell and application clustering across these nodes (see Figure A-1). The Deployment Manager contains the master configuration repository and provides a centralized point of cell administration.

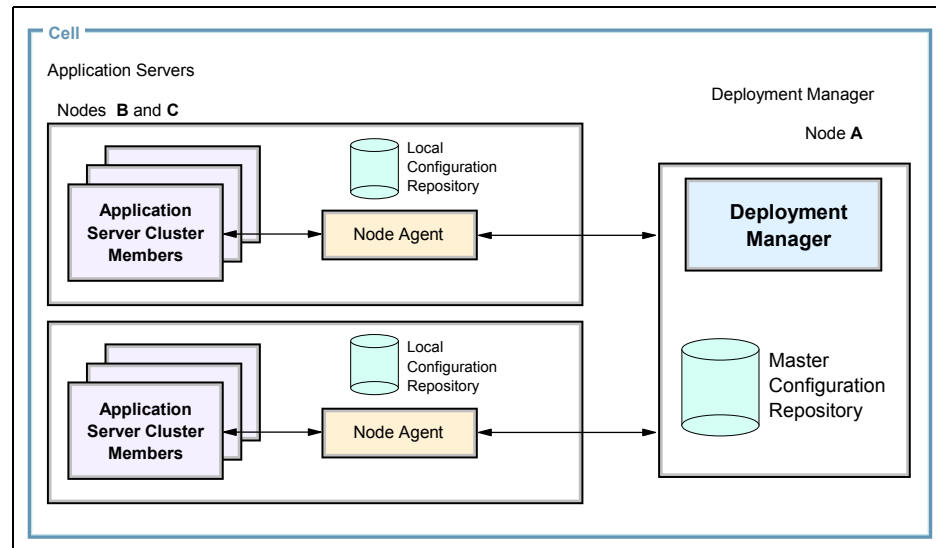


Figure A-1 A Network Deployment cell configuration

In the case of a Base node failure, the other cluster members transparently take over the load of the failed one, and after restoration, the node will continue to participate in sharing the workload again. If the Deployment Manager node fails, the application servers in the cell keep on working, but making configuration changes without the central administration view becomes inefficient and burdensome. You can choose to configure your system in a highly available cluster to minimize downtime, but this would require additional hard- and software and also add to the complexity of the system configuration. For an in-depth discussion of WebSphere Application Server system high availability and failover, refer to Part 4, "High availability solutions" on page 313 in this book.

For an alternative implementation of Deployment Manager high availability *without clustering*, refer to the developerWorks article by Tom Alcott, which provides additionally hints and insights into system backup and restoration:

[http://www.ibm.com/developerworks/apps/transform.wss?URL=/developerworks/websphere/library/techarticles/0304\\_alcott/alcott.xml&xslURL=/developerworks/websphere/xsl/document.xsl&format=one-column](http://www.ibm.com/developerworks/apps/transform.wss?URL=/developerworks/websphere/library/techarticles/0304_alcott/alcott.xml&xslURL=/developerworks/websphere/xsl/document.xsl&format=one-column)

## Backup methods

There are several different methods to backup and restore a WebSphere Application Server node; one of them is to use the **backupConfig** and **restoreConfig** command-line tools that come with the product. Another one is to save the files inside the <WAS\_HOME> directory tree using backup software like IBM Tivoli Storage Manager. Example A-1 shows the output when backing up a repository using the **backupConfig** command.

*Example: A-1 Backing up a repository using backupConfig.bat*

---

```
C:\WebSphere\DeploymentManager\bin>backupConfig.bat dm_2003-11-28.zip -nostop
ADMU0116I: Tool information is being logged in file
          C:\WebSphere\DeploymentManager\logs\backupConfig.log
ADMU5001I: Backing up config directory C:\WebSphere\DeploymentManager\config to
          file C:\WebSphere\DeploymentManager\bin\dm_2003-11-28.zip
.....
.....
ADMU5002I: 150 files successfully backed up

C:\WebSphere\DeploymentManager\bin>
C:\WebSphere\DeploymentManager\bin>dir dm*.zip

12/08/2003  02:43p             16,267,929 dm_2003-11-28.zip
               1 File(s)          16,267,929 bytes
               0 Dir(s)  18,773,573,632 bytes free
```

---

Independent of the backup strategy you implemented, it is recommended to take a backup before major configuration changes or updates, and having backups scheduled on a regular basis: on UNIX platforms, schedule backup jobs using the **cron** utility; on the Windows platform, use the Scheduler facility, or use your backup software's proprietary mechanisms to achieve the same.

**Tip:** The **backupConfig** (and also the **restoreConfig**) command takes the **-nostop** option which allows you to perform a backup in a production environment without stopping any service.

In this appendix we solely focus on the steps necessary to restore a failed system in a *Network Deployment configuration*. Restoring a base configuration, that is, a standalone node, is straightforward and is not discussed here. Refer to chapter 10.7.5 in *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195 for more information on backing up and restoring a standalone base node installation.

## Node failure scenarios

We discuss the following two node failure scenarios in a Network Deployment cell configuration:

1. Failure of the Deployment Manager node
2. Failure of a WebSphere Application Server Base node

### Failure of the Deployment Manager node

As the Deployment Manager provides a centralized view for the administration of the cell, backing up the master cell configuration repository on a regular basis is an important task. If the Deployment Manager fails, the cell's application servers continue to do their work, but you are unable to perform any configuration changes using the Deployment Manager's Administrative Console. It is possible to administer individual application server nodes using **wsadmin**, but any local changes will be overwritten once the Deployment Manager is online again and an automatic or manual repository synchronization is done.

Additional techniques for improving system availability like Deployment Manager high availability through clustering can be found in Part 4, "High availability solutions" on page 313, and also in the white paper by Wang and Bransford *Server Clusters For High Availability in WebSphere Application Server Network Deployment Edition 5.0*, to be found at:

<http://www.ibm.com/support/docview.wss?uid=swg27002473>

### Failure of a WebSphere Application Server Base node

A WebSphere Application Server Base node holds a partial copy of the master cell configuration repository. When it fails, its configuration state is still available in the master copy on the Deployment Manager node, making the restoration process very easy.

Follow these steps to check for an unavailable node:

1. Log on to the WebSphere Administrative Console and select **System Administration -> Node Agents**. The Node Agent on the failed node should

be unavailable because the Deployment Manager has lost contact (see Figure A-2).

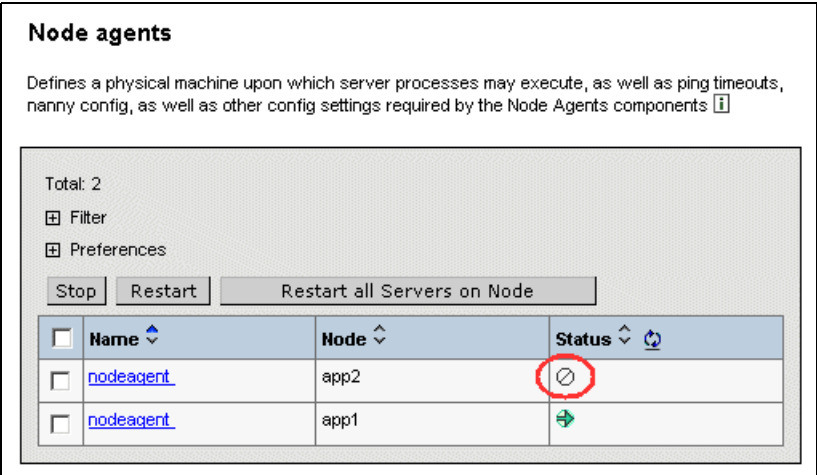


Figure A-2 Status of failed Node Agent (unavailable)

2. Select **System Administration -> Nodes**. In Figure A-3 you can see that the status of the failed node is *Unknown*.

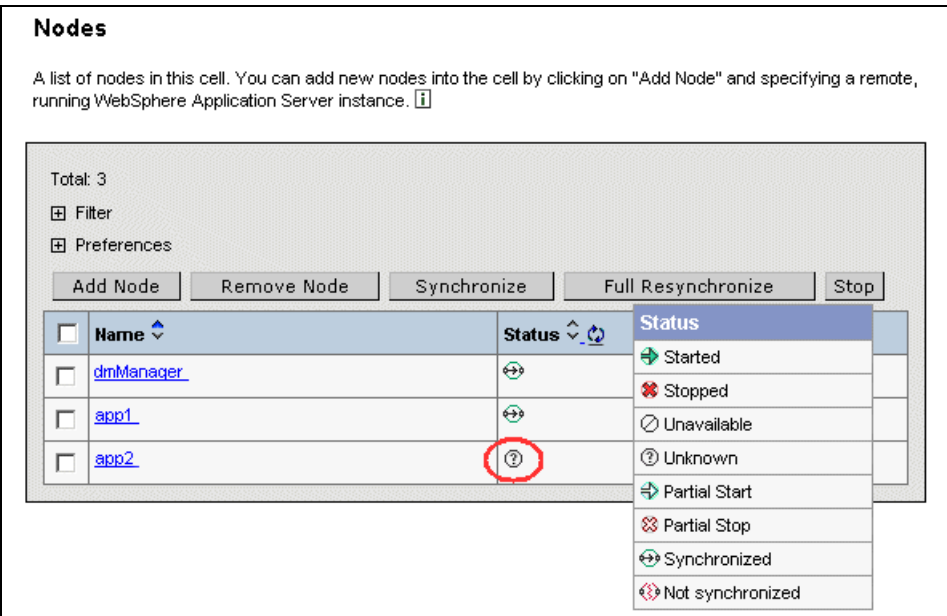


Figure A-3 Display node status for failed node

## Node recovery

The steps necessary for recovering a failed node, either the Deployment Manager or a base node, are explained by using two different methods in this section. The decision which method to use depends on individual requirements and general conditions of your environment, for example, backup storage cost, budget, software licenses for backup software, mean time to repair, service level agreements, etc. These recovery methods are as follows:

1. Using a conventional filesystem backup
2. Using the **backupConfig** and **restoreConfig** command-line tools

**Note:** The recommended way for system recovery is to use the first method, a filesystem based backup and recovery. This recommendation is based on the following two reasons:

- ▶ The **addNode** command used in the **backupConfig** recovery method does not work in versions prior to WebSphere V5.1.
- ▶ The **backupConfig** method does not address SSL keyring files and other property files, while a filesystem backup of <WAS\_HOME> includes these important files.

We assume that the failed node has been either repaired or replaced up to the point where the IBM WebSphere Application Server Network Deployment V5.1 prerequisites are met again (that is, hardware requirements are met, the operating system and all required OS fixes are installed). See the WebSphere Application Server V5.1 hard- and software requirements at:

<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>

Regardless of the recovery method, we always start with a re-install of WebSphere Application Server V5.1 so that the system registry is set up correctly.

**Tip:** Consider installing the software using response files to avoid errors during the (repeated) product and fixes installation and for a quick and silent (unattended) setup process. For detailed information about silent product installation, refer to Chapter 6.4. in *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195 or to the WebSphere V5.1 InfoCenter.



## Recovery using filesystem backup and restore methods

The advantages of using a file system backup and restore over the backupConfig method are that fixes and fixpacks, and the entire configuration including the federated state of the node are restored instantly, shortening node repair time. Although we could restore the filesystem backup only, we decided to initially install the WebSphere Application Server software so that all necessary entries and information in the system registry are set. Disadvantages are that an up-to-date backup for *every* node in the cell is needed, and more files than necessary are backed up, increasing the storage space requirements on your archiving system. Table A-1 lists these advantages and disadvantages:

Table A-1 (Dis-)advantages of using filesystem backup and recovery

Advantages	Disadvantages
<ul style="list-style-type: none"><li>▶ Shorter time to recover a failed node needed</li></ul>	<ul style="list-style-type: none"><li>▶ Every node in the cell has to be backed up</li><li>▶ More space needed on archiving system</li></ul>

### Prerequisites

For the complete recovery of a failed node the following prerequisites have to be met:

- ▶ The installation files for IBM WebSphere Application Server Network Deployment V5.1 are available.
- ▶ An up-to-date filesystem backup of the <WAS\_HOME> subdirectory of the failed node, which is part of a Network Deployment cell, is available.

### Restoring the Deployment Manager node

1. Reinstall IBM WebSphere Application Server Network Deployment V5.1 with the same parameters/features as used at the time of the initial system setup. The setup process will initially configure an empty cell without any federated nodes or application servers.

For information on installing WebSphere Application Server Network Deployment V5.0 and higher, refer to the redbook *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195 or to the WebSphere V5.1 InfoCenter.

2. If the Deployment Manager process is running after installation, stop it using the **stopManager** command.
3. Restore the filesystem backup of the <WAS\_HOME> directory.

**Important:** Remove (or rename) the <WAS\_HOME> directory before restoring the backup. Otherwise this would result in a garbled repository, containing data of the initial default and the newly restored configuration.

4. Start the Deployment Manager using the **startManager** command.  
Check the status of the nodes using the Administrative Console. You should find all previously defined nodes, Node Agents, application servers, enterprise applications, etc.
5. Optionally update the cell configuration with the latest changes since the backup snapshot was taken. Save and synchronize the configuration. Any changes made to the individual application servers will be lost because they are overwritten by the master configuration data from the Deployment Manage node.

### Restoring a WebSphere Application Server Base node

Because all configuration data is kept in the master repository on the Deployment Manager node, the following procedure is fairly easy due to IBM WebSphere Application Server's centralized configuration replication mechanism:

1. Reinstall IBM WebSphere Application Server Base V5.1 (and IBM HTTP Server if collocated on the same system) with the same parameters/features as used at the time of the initial system setup.

The node name of the failed node can be retrieved from the WebSphere Administrative Console by expanding **System Administration** and clicking **Nodes** (see Figure A-3 on page 919).

For information on installing WebSphere Application Server V5.0 and higher and IBM HTTP Server, refer to redbook *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195 or to the WebSphere V5.1 InfoCenter.

2. Restore the filesystem backup into the <WAS\_HOME> directory.

**Important:** Remove (or rename) the <WAS\_HOME> directory before restoring the backup. Otherwise the backup results in a repository containing data of the initial, default and the newly restored configuration.

3. Perform a full node synchronization using the **syncNode** command. The Node Agent contacts the Deployment Manager and performs a full repository synchronization. When it terminates, the node will have the latest configuration data.

For detailed instructions on how to use `syncNode`, refer to Appendix A.7 in *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195 or to the WebSphere V5.1 InfoCenter.

4. Start the Node Agent using the **startNode** command.
5. Start the application server or cluster member on the newly restored node.

## Recovery using `backupConfig` and `restoreConfig`

The **backupConfig** and **restoreConfig** command line utilities are part of WebSphere Application Server with the purpose of providing an easy way of backup and recovery of the configuration repository in case of a node outage.

The advantage of using these tools is that only one backup is needed for the entire cell. When the master configuration data on the Deployment Manager (DM) node is saved with **backupConfig**, a failed Deployment Manager node can easily be restored from that backup. Any failures of the base nodes in that cell can be recovered by using WebSphere Application Server's configuration synchronization mechanism. One drawback of this method is that all previously installed e-fixes and fixpacks have to be reinstalled manually (in contrary to a filesystem restore), but this task can be sped up by using semi-automated, silent installations using response files.

Another disadvantage is that not all necessary files are backed up and restored by the command-line tools supplied with WebSphere: SSL keyring files (default location `<WAS_HOME>/etc`) and the `<WAS_HOME>/property` directory, where, for example, additional security related information is stored, are not saved! Therefore you should only use this method if a filesystem based recovery approach is not practicable, or if you are able to use the default, initially installed versions of the above mentioned files. The advantages and disadvantages of this approach are listed in Table A-2.

For detailed instructions on how to use **backupConfig** and **restoreConfig**, refer to appendix A.14 and A.15 in *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195 or to the WebSphere V5.1 InfoCenter.

Table A-2 (Dis-)advantages of using `backupConfig` and `restoreConfig` for recovery

Advantages	Disadvantages
<ul style="list-style-type: none"> <li>▶ Only one backup per cell is needed</li> <li>▶ Takes full advantage of distributed repository synchronization</li> </ul>	<ul style="list-style-type: none"> <li>▶ e-fixes and fixpacks have to be installed manually</li> <li>▶ Several additional files have to be copied/restored manually</li> </ul>

## Prerequisites

For the complete recovery of a failed node the following prerequisites have to be fulfilled:

- ▶ The installation files for IBM WebSphere Application Server Network Deployment V5.1 are available.
- ▶ All previously installed fixes and fixpacks are available.
- ▶ A previously saved configuration backup file as produced by the **backupConfig** command is available. The default filename of a backup archive is `<WAS_HOME>/bin/WebSphereConfig_YYYY-MM-DD.zip`.
- ▶ Additionally, some properties files are not restored, because they are located outside the `<WAS_HOME>/config` directory subtree. In a cluster configuration these files are identical to the ones on the other cluster members in the cell. Depending on whether you adapted these files, you either have to repeat that configuration step, or to copy them manually from another cluster member or the Deployment Manager to the restored node.

These files are as follows (not necessarily a complete list):

- Property files, for example, security related files in the `<WAS_HOME>/properties/` directory:
  - `sas.client.properties`
  - `sas.server.properties`
  - `wsadmin.properties`
  - ...
- Files in the `<WAS_HOME>/etc/` directory, for example, SSL keyring files like `plugin-key.kdb`, etc.
- The plug-in configuration file, if it was manually modified on the Web server node(s).

## Restoring the Deployment Manager node

1. Reinstall IBM WebSphere Application Server Network Deployment V5.1 with the same parameters/features as used at the time of the initial system setup. Specify the same values for *node name*, *host name* and *cell name* in the setup dialog as during the previous installation. The setup process initially configures an empty cell without any federated nodes or application servers.

For information on installing WebSphere Application Server Network Deployment V5.0 and higher, refer to redbook *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195 or to the WebSphere V5.1 InfoCenter.

2. If the Deployment Manager process is running after installation, stop it using the **stopManager** command.

3. Reinstall all fixes and fixpacks that were previously installed on the system using the *Updateinstaller* product. The Updateinstaller comes with fixpacks, but can also be obtained separately from the WebSphere support page at

<http://www.ibm.com/software/webservers/appserv/was/support/>

by selecting **Updateinstaller** in the **Self Help -> Download** section. Refer to the included readme file and documentation on how to use the Updateinstaller program.

**Attention:** The Deployment Manager must have the highest fix level in the entire cell!

4. Restore a previously taken configuration backup using the **restoreConfig** command (see Example A-2). The initial cell configuration data will automatically be backed up and replaced by the restored configuration in this step.

*Example: A-2 Restoring a configuration using restoreConfig.bat*

---

```
C:\WebSphere\DeploymentManager\bin>restoreConfig.bat dm_backup_2003-11-28.zip
ADMU0116I: Tool information is being logged in file
           C:\WebSphere\DeploymentManager\logs\restoreConfig.log
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: dmgr
ADMU2010I: Stopping all server processes for node dmManager
ADMU0510I: Server dmgr is now STOPPED
ADMU5502I: The directory C:\WebSphere\DeploymentManager\config already exists;
           renaming to C:\WebSphere\DeploymentManager\config.old
ADMU5504I: Restore location successfully renamed
ADMU5505I: Restoring file dm_backup_2003-11-28.zip to location
           C:\WebSphere\DeploymentManager\config
.....
ADMU5506I: 138 files successfully restored
ADMA6001I: Begin App Preparation -
ADMA6009I: Processing complete.
```

---

5. Check the status of the nodes using the Administrative Console: you should find all previously defined nodes, Node Agents, application servers, enterprise applications, etc.
6. Start the Deployment Manager by using the **startManager** command.
7. Optionally update the cell configuration with the latest changes since the backup snapshot was taken. Save and synchronize the configuration. Any changes made to the individual application servers will be lost because they

are overwritten by the master configuration data from the Deployment Manager node.

## Restoring a WebSphere Application Server Base node

Because all configuration data is kept in the master repository on the Deployment Manager node, the following procedure is fairly easy due to WebSphere Application Server's centralized configuration replication mechanism.

**Important:** The hereafter described method, especially the **addNode** command, may not work for versions prior to WebSphere Application Server V5.1!

1. Reinstall IBM WebSphere Application Server Base V5.1 (and IBM HTTP Server if collocated on the same system) with the same parameters/features as before. Specify the same values for *node name* and *host name* in the setup dialog as during the previous installation.

The node name of the failed node can be retrieved from the WebSphere Administrative Console by expanding **System Administration** and clicking **Nodes** (see Figure A-3 on page 919).

**Note:** It is important to use exactly the same node name during reinstallation of IBM WebSphere Application Server Base V5.1. Otherwise the repository synchronization will fail to restore the node's previous configuration state, simply because the node is still unavailable from the Deployment Manager's point of view. Instead, another *new* node will be added to the cell after node federation, which then needs to be configured manually.

For information on installing WebSphere Application Server V5.0 and higher and IBM HTTP Server, refer to the redbook *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195 or to the WebSphere V5.1 InfoCenter.

2. Reinstall all fixes and fixpacks that were previously installed on the system using the Updateinstaller. The Updateinstaller comes with fixpacks, but can also be obtained separately from the WebSphere support page found at

<http://www.ibm.com/software/webservers/appserv/was/support/>

by clicking the **Updateinstaller** link in the **Self Help -> Download** section. Refer to the included readme file and documentation on how to use the Updateinstaller program.

3. Federate the base node into the cell using the **addNode** command or the WebSphere Administrative Console. For detailed instructions on how to federate a node into a cell, refer to chapter 7.4.4 in *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195 or to the WebSphere V5.1 InfoCenter.

This step will automatically start the Node Agent on the base node, who in turn performs an initial repository synchronization; after that the newly restored node has the latest configuration data available.

4. Start the application server or cluster member on the newly restored node.

## Conclusion

Using either filesystem backups or taking configuration snapshots by running the backupConfig command on a regular basis lets you easily restore a failed node in a Network Deployment configuration. If downtime is critical, stick to the filesystem recovery method. For a small cell, when the focus is centered more around the budget and additional files like SSL keyring files or property files are no issue, WebSphere Application Server's command-line tools backupConfig and restoreConfig provide another, cheaper, but nearly equally effective alternative.

## Additional reference material

These documents contain additional information on backing up and restoring WebSphere Application Server V5.0 and higher:

- ▶ Part 4, "High availability solutions" on page 313.
- ▶ Chapter 10.7., "Common system management tasks" in *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195.
- ▶ Appendix A., "Command-line tools" in *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195.
- ▶ *Implementing a Highly Available Infrastructure for WebSphere Application Server Network Deployment, Version 5.0 without Clustering* at:

[http://www.ibm.com/developerworks/apps/transform.wss?URL=/developerworks/websphere/library/techarticles/0304\\_alcott/alcott.xml&xslURL=/developerworks/websphere/xsl/document.xsl&format=one-column](http://www.ibm.com/developerworks/apps/transform.wss?URL=/developerworks/websphere/library/techarticles/0304_alcott/alcott.xml&xslURL=/developerworks/websphere/xsl/document.xsl&format=one-column)

- White paper *Server Clusters for High Availability in WebSphere Application Server Network Deployment Edition 5.0* at:

<http://www.ibm.com/support/docview.wss?uid=swg27002473>





# B

## **Sample URL rewrite servlet**

This appendix describes how to set up an example to test session management using URL rewrites.

## Setting up the servlet

The class `SessionSampleURLRewrite` has been provided since none of the standard samples use URL rewriting to manage sessions. This is because using URL rewrites requires additions to the normal servlet code.

## Source code

Example B-1 lists the Java source for the class. You can either compile and package this class yourself or you can use the pre-compiled class file packaged in `urctest.war`. You can find information on how to download `urctest.war` in Appendix C, “Additional material” on page 935.

*Example: B-1 SessionSampleURLRewrite class source code*

---

```
public class SessionSampleURLRewrite extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // Step 1: Get the Session object

        boolean create = true;
        HttpSession session = request.getSession(create);

        // Step 2: Get the session data value

        Integer ival = (Integer)
            session.getAttribute ("sessiontest.counter");
        if (ival == null) ival = new Integer (1);
        else ival = new Integer (ival.intValue () + 1);
        session.setAttribute ("sessiontest.counter", ival);

        // Step 3: Rewrite the session ID onto the URL

        String contextPath = request.getContextPath();
        String servletName = request.getServletPath();
        String encodeString = contextPath + servletName;
        String encodedURL = response.encodeURL(encodeString);

        // Step 4: Output the page

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Session Tracking Test</title></head>");
    }
}
```

```

        out.println("<body>");
        out.println("<h1>Session Tracking Test</h1>");
        out.println ("You have hit this page " + ival + " times" + "<br>");

        if (!session.isNew()) {
            out.println("<p>");
            if(request.isRequestedSessionIdFromURL())
                out.print("Your requested session ID was found in a rewritten
URL");
            else
                out.print("Your requested session ID was NOT found in a rewritten
URL");
        }

        // Use the rewritten URL as the link, You must enable URL Rewriting in the
        session Manager and
        // disable cookies in either the Session Manager or the browser

        out.print("<p>");
        out.print("<a href=\"");
        out.print(encodedURL);
        out.println("\">Request this servlet again using the rewritten URL</a>");

        out.println("</body></html>");
    }
}

```

---

## Steps to install SessionSampleURLRewrite servlet

We provide two alternatives to install the SessionSampleURLRewrite servlet:

- ▶ Installing the urltest Web module
- ▶ Adding the servlet to an installed application

## Installing the urltest Web module

Please refer to 7.6.2, “Install BeenThere” on page 290 on detailed instructions about how to install an enterprise application. In summary, these are the steps to install urltest.war:

1. Locate urltest.war as described in Appendix C, “Additional material” on page 935 and store it on your workstation or on your server.
2. Open the WebSphere Administrative Console and select **Application -> New Application**.
3. In the Preparing for the application installation window:
  - a. Browse to the urltest.war Web module archive (Local path or Server path).
  - b. Enter a context root, such as /urltest.
  - c. Click **Next**.
4. Leave all defaults on the Preparing for the application installation window and click **Next** again.
5. If desired, change the name of the application (default = urltest\_war) in Step 1 of the Install New Application procedure. Click **Next**.
6. Map the Web Module urltest to the desired Virtual Host. Clicking **Next** brings you to Step 3.
7. On the Step 3: Map modules to application servers window, from the Clusters and Servers selection box choose the application server you wish to install urltest.war on. Then check the box for the module urltest and click **Apply**. Click **Next** once again.
8. Verify the installation on the Step 4: Summary window and click **Finish** to install the urltest Web module.
9. Save the changes to the master configuration.
10. To start the urltest\_war enterprise application, locate urltest\_war in the Enterprise Applications view. Select the application and click **Start**.
11. Regenerate the plug-in configuration: Select **Environment -> Update Web Server Plug-in**. Click the **OK** button to update the plug-in configuration file.
12. Change the path information in <WAS\_HOME>/config/plugin-cfg.xml file, then copy it to your remote Web servers (if needed, refer to 7.6.3, “Regenerate Web server plug-in” on page 294 for instructions on how to do so).

## Adding the servlet to an installed application

**Note:** While convenient in a development environment, the following technique is not recommended in a production environment.

Follow these steps to install the servlet into the sample application. This procedure can be performed whether the application server is running or not. If the application server is running, then the changes will be picked up automatically, subject to the auto-reload settings on your Web container. If auto-reload is disabled, the servlet will not be available until the application server is restarted.

This procedure should be repeated for each of the cluster members of the cluster you are using to test session management.

1. Compile or download the SessionSampleURLRewrite.class file.
2. Place the SessionSampleURLRewrite.class file in the directory:

`<WAS_HOME>/installedApps/<node_name>/DefaultApplication.ear/Default  
WebApplication.war/WEB_INF/classes`

For example:

`c:\websphere\appserver\installedApps\app1\DefaultApplication.ear\Default  
WebApplication.war\WEB_INF\classes`

3. Open the web.xml file for editing within the directory:  
`<WAS_HOME>/config/cells/<node_name>/applications/DefaultApplication.  
ear/deployments/DefaultApplication/DefaultWebApplication.war/WEB-INF`
4. Locate the last servlet tag definition, just before the first servlet-mapping tag, and add the lines listed in Example B-2 at this point.

Make sure that the IDs used follow the pattern in the file; for example the last servlet definition should be Servlet\_3, so use Servlet\_4 for your new servlet definition.

*Example: B-2 Lines to add to web.xml file*

---

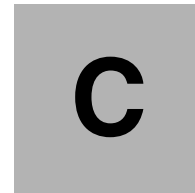
```
<servlet id="Servlet_4">  
  <servlet-name>urltest</servlet-name>  
  <description>Test URL rewrite</description>  
  <servlet-class>SessionSampleURLRewrite</servlet-class>  
</servlet>  
<servlet-mapping id="ServletMapping_4">  
  <servlet-name>urltest</servlet-name>  
  <url-pattern>/servlet/urltest</url-pattern>  
</servlet-mapping>
```

---

5. Save and close the file.
6. If the application server is not running, start the application server. If it is already running but you have disabled auto-reload (See “Auto-reload setting of your Web module” on page 152), restart the application server.
7. Regenerate the plug-in configuration file for your Web server(s). See “Manual regeneration of the plug-in file” on page 153 to see how to do this.
8. Open the regenerated plugin.cfg.xml file and verify that the URL we defined in step 4 has been entered. The following line should appear:

```
<Uri Name="/servlet/urltest"/>
```

Repeat these steps for each of the nodes in your server group. Once this is complete, you will be able to run the test shown in “URL rewriting” on page 194.



# Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

## Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246198>

Alternatively, you can go to the IBM Redbooks Web site at:

[ibm.com/redbooks](http://ibm.com/redbooks)

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG24-6198-01.

## Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
<b>urltest.war</b>	URL Rewrite Example Web module
<b>morton0206.zip</b>	BeenThere application code
<b>TradeRedirector.zip</b>	Trade3 back-end application for WebSphere MQ scenario

## System requirements for downloading the Web material

The following system configuration is recommended:

<b>Hard disk space:</b>	10 MB
<b>Operating System:</b>	Windows, AIX, Solaris, Linux, OS/400®
<b>Processor:</b>	500 MHz Pentium®, RS/6000®, iSeries, Sparc
<b>Memory:</b>	384 MB RAM minimum

## How to use the Web material

- ▶ Create a subdirectory (folder) on your workstation or your server, and download the urltest.war file into this folder.
- ▶ Create a subdirectory (folder) on a Windows system and download the morton0206.zip file into this folder. Extract the BeenThere50.ear file from the zip file and copy it either into the <install\_root>/installableApps directory of your application server or to a local folder on your workstation. This Enterprise Application Resource is all you need to install the BeenThere application on your applications server(s).
- ▶ Download TradeRedirector.zip. The zip-file consists of two files:
  - TradeRedirector.ear
  - Instructions for setup of TradeRedirector.txtFollow the install instructions from the text file.



# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

## IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 945. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *IBM WebSphere Application Server V5.1 System Management and Configuration*, SG24-6195
- ▶ *IBM WebSphere V5.0 Security*, SG24-6573
- ▶ *WebSphere Edge Server: Working with Web Traffic Express and Network Dispatcher*, SG24-6172
- ▶ *Patterns for the Edge of Network*, SG24-6822
- ▶ *Migration to WebSphere Business Integration Message Broker V5*, SG24-6995
- ▶ *EJB 2.0 Development with WebSphere Studio Application Developer*, SG24-6819
- ▶ *Patterns: Self-Service Application Solutions Using WebSphere V5.0*, SG24-6591

## Other publications

These publications are also relevant as further information sources:

- ▶ *Load Balancer Administration Guide*, GC09-4602  
<http://www.ibm.com/software/webservers/appserv/doc/v50/ec/infocenter/index.html>
- ▶ *Caching Proxy Administration Guide*, GC09-4601  
<http://www.ibm.com/software/webservers/appserv/doc/v50/ec/infocenter/index.html>

- ▶ *HACMP for AIX Installation Guide*  
[http://www-1.ibm.com/servers/eserver/pseries/library/hacmp\\_docs.html](http://www-1.ibm.com/servers/eserver/pseries/library/hacmp_docs.html)
- ▶ *WebSphere MQ using Java*  
<http://www.ibm.com/software/integration/mqfamily/library/manualsa/manuals/crosslatest.html>
- ▶ *WebSphere Application Server Development Best Practices for Performance and Scalability*, by Harvey W. Gunther, IBM, 2000  
[http://www.ibm.com/software/webservers/appserv/ws\\_bestpractices.pdf](http://www.ibm.com/software/webservers/appserv/ws_bestpractices.pdf)
- ▶ IBM JVM Diagnostics Guides  
<http://www.ibm.com/developerworks/java/jdk/diagnosis/>
- ▶ “Performance Testing Protocol for WebSphere Application Server-based Applications”  
[http://www7b.software.ibm.com/wsdd/techjournal/0211\\_polozoff/polozoff.html](http://www7b.software.ibm.com/wsdd/techjournal/0211_polozoff/polozoff.html)
- ▶ *Performance Analysis for Java Web Sites*, by Stacy Joines, et.al. Addison-Wesley, September 2002, ISBN 0201844540.

## Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ WebSphere Application Server InfoCenter  
<http://www.ibm.com/software/webservers/appserv/infocenter.html>
- ▶ WebSphere V5.1 InfoCenter  
<http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp>
- ▶ WebSphere Application Server 5.1 InfoCenter messaging tuning section  
[http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tprf\\_tunejms.html](http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tprf_tunejms.html)
- ▶ Content related to dynamic caching in the WebSphere InfoCenter  
[http://publib.boulder.ibm.com/infocenter/wasinfo/topic/com.ibm.websphere.nd.doc/info/ae/ae/tprf\\_dynamiccache.html](http://publib.boulder.ibm.com/infocenter/wasinfo/topic/com.ibm.websphere.nd.doc/info/ae/ae/tprf_dynamiccache.html)
- ▶ Edge Components InfoCenter  
<http://www.ibm.com/software/webservers/appserv/doc/v51/ec/infocenter/index.html>
- ▶ IBM WebSphere Application Server  
<http://www.ibm.com/software/webservers/appserv>

- ▶ IBM WebSphere Application Server hardware and software requirements  
<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>
- ▶ IBM WebSphere Edge Server  
<http://www.ibm.com/software/webservers/edgeserver>
- ▶ IBM DB2 UDB hardware and software requirements  
<http://www.ibm.com/software/data/db2/udb/sysreqs.html>
- ▶ HTTP server configuration at IBM HTTP Server InfoCenter  
<http://www.ibm.com/software/webservers/httpservers/library.html>
- ▶ AIX Fix Distribution Service  
<http://techsupport.services.ibm.com/rs6k/fixdb.html>
- ▶ IBM WebSphere Application Server Library  
<http://www.ibm.com/software/webservers/appserv/library/index.html>
- ▶ WebSphere Application Server Development white papers  
<http://www.ibm.com/software/webservers/appserv/whitepapers.html>
- ▶ WebSphere Best Practices Web site  
<http://www7b.software.ibm.com/wsdd/zones/bp/>
- ▶ WebSphere Application Server zone at WebSphere Developer Domain  
<http://www7b.boulder.ibm.com/wsdd/zones/was/>
- ▶ WebSphere performance Web site, including Trade3 code download  
<http://www.ibm.com/software/webservers/appserv/was/performance.html>
- ▶ BeenThere installation description and link to download  
<http://www.sys-con.com/websphere/article.cfm?id=321>
- ▶ IBM @server iSeries performance documents  
<http://www.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/product/PerformanceConsiderations.html>
- ▶ Updateinstaller download  
<http://www.ibm.com/software/webservers/appserv/was/support/>
- ▶ IHS 2.0 download  
<http://www.ibm.com/software/webservers/httpservers/>
- ▶ GUI tool for building cache policy files  
<http://www.alphaworks.ibm.com/tech/cachepolicyeditor>

- ▶ ASTK download for WebSphere V5.0.2 the ASTK  
[http://www.ibm.com/support/docview.wss?rs=180&context=SSEQTP&q=&uid=swg24005125&loc=en\\_US&cs=utf-8&lang=en+en](http://www.ibm.com/support/docview.wss?rs=180&context=SSEQTP&q=&uid=swg24005125&loc=en_US&cs=utf-8&lang=en+en)
- ▶ WebSphere Developer Domain  
<http://www7b.software.ibm.com/wsdd/>
- ▶ ThreadAnalyzer, Performance Advisors, and Assembly Toolkit for WebSphere Application Server code download  
<http://www.ibm.com/developerworks/websphere/downloads/techpreviews.html>
- ▶ IBM developerWorks  
<http://www.ibm.com/developerworks/>
- ▶ IBM developerWorks WebSphere Best Practices Zone  
<http://www.ibm.com/developerworks/websphere/zones/bp/background.html>  
<http://www.ibm.com/developerworks/websphere/zones/bp/index.html>
- ▶ IBM alphaWorks  
<http://www.alphaworks.ibm.com>
- ▶ Page Detailer information and download  
<http://www.alphaworks.ibm.com/tech/pagedetailer>
- ▶ IBM WebSphere Studio Workload Simulator  
<http://www.ibm.com/software/awdtools/studioworkloadsimulator/>
- ▶ IBM Tivoli Monitoring for Web Infrastructure  
<http://www.ibm.com/software/tivoli/products/monitor-web/>
- ▶ IBM Tivoli Monitoring for Transaction Performance  
<http://www.ibm.com/software/tivoli/products/monitor-transaction/>
- ▶ List of IBM's Business Partners that offer performance monitoring tools compliant with WebSphere Application Server  
[http://www.ibm.com/software/webservers/pw/dhtml/wsperformance/performance\\_bpsolutions.html](http://www.ibm.com/software/webservers/pw/dhtml/wsperformance/performance_bpsolutions.html)
- ▶ Java Performance Tuning Web site  
<http://www.javaperformancetuning.com/>
- ▶ Supportpacs about JMS performance with WebSphere MQ  
[http://www-1.ibm.com/support/docview.wss?rs=203&uid=swg24006854&loc=en\\_US&cs=utf-8&lang=en](http://www-1.ibm.com/support/docview.wss?rs=203&uid=swg24006854&loc=en_US&cs=utf-8&lang=en)  
  
[http://www-1.ibm.com/support/docview.wss?rs=203&uid=swg24006902&loc=en\\_US&cs=utf-8&lang=en](http://www-1.ibm.com/support/docview.wss?rs=203&uid=swg24006902&loc=en_US&cs=utf-8&lang=en)

- ▶ Design for Scalability - An Update  
<http://www.ibm.com/developerworks/websphere/library/techarticles/hvws/scalability.html>
- ▶ Caching Technologies for Web Applications  
<http://www.almaden.ibm.com/u/mohan/>
- ▶ Exploiting Dynamic Caching in WAS 5.0, Part 1 & 2  
<http://www.e-promag.com/eparchive//index.cfm?fuseaction=viewarticle&ContentID=3623&publicationid=19&PageView=Search&channel=2>  
<http://www.e-promag.com/eparchive//index.cfm?fuseaction=viewarticle&ContentID=4409&publicationid=19&PageView=Search&channel=2>
- ▶ Monitoring Performance with WebSphere  
<http://www.e-promag.com/eparchive//index.cfm?fuseaction=viewarticle&ContentID=1492&publicationid=13&PageView=Search&channel=2>
- ▶ Hints on Running a High-Performance Web Server - Tuning IBM IHS  
<http://www.ibm.com/software/webserver/httpserver/doc/v136/misc/perf.html>
- ▶ Enhancements and Changes in J2SE 1.4.1 Platform  
[http://java.sun.com/products/archive/j2se/1.4.1\\_07/changes.html](http://java.sun.com/products/archive/j2se/1.4.1_07/changes.html)
- ▶ Reducing Garbage Collection Times and Sizing Memory  
<http://developers.sun.com/techtopics/mobility/midp/articles/garbage/>
- ▶ Improving Java Application Performance and Scalability by Reducing Garbage Collection Times and Sizing Memory Using JDK 1.4.1  
<http://developers.sun.com/techtopics/mobility/midp/articles/garbagecollection2/>
- ▶ Fine-tuning Java garbage collection performance  
<http://www.ibm.com/developerworks/ibm/library/i-gctroub/>
- ▶ Garbage collection in the 1.4.1 JVM  
<http://www.ibm.com/developerworks/java/library/j-jtp11253/>
- ▶ A brief history of garbage collection  
<http://www.ibm.com/developerworks/java/library/j-jtp10283/>
- ▶ Sensible Sanitation: Understanding the IBM Java Garbage Collection, Parts 1 and 2  
<http://www.ibm.com/developerworks/ibm/library/i-garbage1/>  
<http://www.ibm.com/developerworks/ibm/library/i-garbage2/>

- ▶ Sensible Sanitation: Understanding the IBM Java Garbage Collection Part 3: verbosegc and command-line parameters  
<http://www.ibm.com/developerworks/library/i-garbage3.html>
- ▶ J2EE Application Development: One or many applications per application server?  
[http://www7b.software.ibm.com/wsdd/techjournal/0211\\_alcott/alcott.html](http://www7b.software.ibm.com/wsdd/techjournal/0211_alcott/alcott.html)
- ▶ Handling Static content in WebSphere Application Server  
[http://www7b.software.ibm.com/wsdd/techjournal/0211\\_brown/brown.html](http://www7b.software.ibm.com/wsdd/techjournal/0211_brown/brown.html)
- ▶ Willy Chiu, *Design for Scalability - An Update*, IBM, 2001  
<http://www7b.boulder.ibm.com/wsdd/library/techarticles/hvws/scalability.html>
- ▶ IBM WebSphere Developer Technical Journal: Writing PMI applications using the JMX interface  
[http://www.ibm.com/developerworks/websphere/techjournal/0402\\_qiao/0402\\_qiao.html](http://www.ibm.com/developerworks/websphere/techjournal/0402_qiao/0402_qiao.html)
- ▶ IBM WebSphere Developer Technical Journal: Writing a Performance Monitoring Tool Using WebSphere Application Server's Performance Monitoring Infrastructure API  
[http://www.ibm.com/developerworks/websphere/techjournal/0202\\_rangaswamy/rangaswamy.html](http://www.ibm.com/developerworks/websphere/techjournal/0202_rangaswamy/rangaswamy.html)
- ▶ WebSphere Performance Diagnostic - Going beyond the Metrics  
<http://www.sys-con.com/websphere/article.cfm?id=207>
- ▶ Design for Performance: Analysis of Download Times for Page Elements Suggests Ways to Optimize  
<http://www.ibm.com/developerworks/websphere/library/techarticles/hvws/performance.html>
- ▶ Mark that trash - Incremental compaction in the IBM JDK Garbage Collector  
<http://www.ibm.com/developerworks/ibm/library/i-incrcmp/>
- ▶ Implementing a Highly Available Infrastructure for WebSphere Application Server Network Deployment, Version 5.0 without Clustering  
[http://www.ibm.com/developerworks/apps/transform.wss?URL=/developerworks/websphere/library/techarticles/0304\\_alcott/alcott.xml&xs1URL=/developerworks/websphere/xsl/document.xsl&format=one-column](http://www.ibm.com/developerworks/apps/transform.wss?URL=/developerworks/websphere/library/techarticles/0304_alcott/alcott.xml&xs1URL=/developerworks/websphere/xsl/document.xsl&format=one-column)
- ▶ Server Clusters For High Availability in WebSphere Application Server Network Deployment Edition 5.0  
<http://www.ibm.com/support/docview.wss?uid=swg27002473>

- ▶ JMS Application Architectures  
<http://www.theserverside.com/articles/article.tss?l=JMSArchitecture>
- ▶ JMS Topologies and Configurations with WebSphere Application Server and WebSphere Studio Version 5  
[http://www.ibm.com/developerworks/websphere/library/techarticles/0310\\_barci\\_a/barcia.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0310_barci_a/barcia.html)
- ▶ MSCS step-by-step configuration guide  
<http://microsoft.com/windows2000/techinfo/planning/server/clustersteps.asp>
- ▶ Log4J  
<http://jakarta.apache.org/log4j/>
- ▶ Java Security, JAAS, JCE and JSSE  
<http://java.sun.com/security>
- ▶ Apache Software Foundation  
<http://www.apache.org>
- ▶ Apache HTTP Server source-code  
<http://httpd.apache.org/download.cgi>
- ▶ Apache 2.0 Feature list  
[http://httpd.apache.org/docs-2.0/new\\_features\\_2\\_0.html](http://httpd.apache.org/docs-2.0/new_features_2_0.html)
- ▶ Apache HTTP Server documentation  
<http://httpd.apache.org/docs-2.0/mpm.html>
- ▶ ApacheBench manual and list of options  
<http://httpd.apache.org/docs/programs/ab.html>
- ▶ Microsoft Data Access Components (MDAC) version 2.5 download  
<http://microsoft.com/data/download.htm>
- ▶ Mercury LoadRunner  
<http://www.merc-int.com>
- ▶ Rational Suite TestStudio  
<http://www.ibm.com/software/awdtools/suite/>
- ▶ Radview WebLOAD  
<http://www.radview.com>
- ▶ Mindcraft WebStone  
<http://www.mindcraft.com>

- ▶ Opendemand Systems OpenLoad  
<http://www.opendemand.com/openload/>
- ▶ Akamai  
<http://www.akamai.com>
- ▶ Speedera  
<http://www.speedera.com>
- ▶ Network Appliance  
<http://www.netapp.com>
- ▶ Blue Coat  
<http://www.bluecoat.com>
- ▶ Cisco Cache Engine  
<http://www.cisco.com>
- ▶ InfoLibria DynaCache  
<http://www.infolibria.com>
- ▶ Edge Side Include (ESI)  
<http://www.esi.org>
- ▶ Sun Solaris performance information  
<http://www.sean.de/Solaris/soltune.html>
- ▶ OpenSTA V1.4.2  
<http://www.opensta.org/download.html>
- ▶ OpenSTA community site  
<http://portal.opensta.org/>
- ▶ OpenSTA sourcecode  
<http://opensta.sourceforge.net/>
- ▶ IBM Rational Suite TestStudio  
<http://www.ibm.com/software/awdtools/suite/>
- ▶ JMeter, Open Source software from the Apache Software Foundation  
<http://jakarta.apache.org/jmeter/>
- ▶ TestMaker and TestNetwork, from PushToTest  
<http://www.pushtotest.com/>
- ▶ Grinder  
<http://grinder.sourceforge.net>



- Segue SilkPerformer

<http://www.segue.com/products/load-stress-performance-testing/index.asp>

## How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)



# Index

## Symbols

%CLIENTSAS% 711  
%CLIENTSOAP% 711  
.war file 62

## Numerics

2phase commit 617  
3DES 903  
80/20 rule 804

## A

### AAT

- IBM Extensions 152
- Access intent 815, 863, 867
  - Optimistic read 815
  - Optimistic update 815
  - Pessimistic read 815
  - Pessimistic update 815
  - Pessimistic update - Weakest Lock At Load 815
  - Read 867
- Access log 890
- Access patterns 851
- Access plan 908
- Activate at 864
  - Once 864
  - Transaction 864
- Active cookie 120
- Active Threads 857, 898
- Active/Active mode 318
- Active/Standby mode 318
- ActiveThreads count 899
- activity.log 180, 182
- Adapter Bean Name 569
- addNode 381, 920, 926–927
- admin\_host 137
- Administrative cell 11
- Administrative Console 11
  - Regenerate plug-in 154
- Administrative domain 390
  - Multiple 335
  - Performance 335
- Administrative server process

- Failover 475
- Administrative service 11, 689
- ADV\_was.java 113–114
- Advisor 96, 717
- Affinity
  - Process 230, 233, 240, 243, 246
  - Transaction 218, 233, 240, 243, 246
- afpaplugin.dll 576
- Agent Controller 760, 762–763, 774
- AIX 436
- Akamai 529, 570
- Akamai EdgeSuite 570
- AKstress *See* WPT
- AKtools 823
- Alarm Manager 695
- Alexandre Polozoff 822
- Apache 10, 892, 894
- Apache HTTP Server 824
- Apache JMeter 775, 805
- ApacheBench 110
- Appliance server 46, 51
- Application
  - Execution 762
  - Monitoring 805
  - Monitoring with Tivoli Performance Viewer 716
  - Performance 804
  - Scalability 804
- Application assembly performance 863
- Application Assembly Tool *See* AAT
- Application clients 54
- Application design 610
- Application development
  - Architecture 804
  - Coding issues 817
  - Design 804
  - EJBs 813
  - Enterprise JavaBeans 804
  - Hidden form fields 810
  - Java coding 804
  - JavaServer Pages 804, 811
  - Memory allocation 805
  - Memory deallocation 805
  - Memory usage 805
  - Performance optimization 805

- Performance target 804
- Performance testing 760
- Reclaim memory 805
- Reuse of algorithms 804
- Reuse of patterns 804
- Servlets 804, 811
- Application performance 804
  - Health 873
  - Testing 822
- Application server 10, 33
  - Database failover 476
  - HACMP 317
- Application Server Facilities 621
- Applied (weak) affinity 370
- ARM 687
- ARM Agents 723
- ARM interface 724
- Array 818
- ASF *See* Application Server Facilities
- Asynchronous message processing 816
- Asynchronous JMX messaging 408
- Asynchronous messaging system 593
- Attribute
  - Transient 809
- Authentication level 644
- Automatic deadlock detection 741
- AutoRaid disk array 440
- Auto-reload
  - Web module 151–152
- Auto-reload setting 152
- Availability 4, 6, 61, 67, 69, 94
  - Best possible topologies 333
  - Causes of downtime 325
  - Continuous 323
  - EJB container 80
  - Failover 6
  - Hardware-based high availability 6
  - Levels 319
  - Maintainability 7
  - Single point of failure 61
  - SPOF 61
  - Uptime 522
  - Web container 80
  - Web server 80
- Availability matrix 323
- Average servlet response time 723
- Avg Method RT 691–692

## B

- Back end application 599
- Backout queues 638
- Backout threshold 638
- Backup
  - Filesystem 921
  - Master cell configuration repository 918
  - Network Deployment configuration 915
  - Property files 920
  - Scheduling 917
  - SSL keyring files 920
- Backup cluster 248
  - Bootstrap host 249
  - Fail back 251
- Backup servers 346, 349
- backupConfig 915, 917, 923
- BackupServers tag 163, 349
- Basic systems 321
- Batch 47
- Bean cache 863
- Bean Managed Persistence 814–815, 867
- Beans
  - Passivating 815
- Beans.instantiate() 811
- BeenThere 184, 186, 228, 239
  - Installation 290
- BeenThere application 290
- Benchmark
  - Trade3 259
  - Trade3.1 298
- Big3 427, 439
- bind 908–909
- BINDINGS 644
- BLOB 809
- Blue Coat 529
- BMP *See* Bean Managed Persistence
- Boot address 437
- Bootstrap 220, 468
- Bootstrap server 220
- Bootstrap server provider URL 220
- Borland Optimizelt 808
- Bottleneck 734, 847, 849, 855, 857, 862, 873
  - Garbage collection 881
  - Processing 847
- Boundary statistic 690
- Bounded range statistic 690
- Broker 618
- Broker administrator perspective 663
- BrokerCommandFailedException 645

- browse-parse-get loop 631
- Buffer pool 908
- Bulk encryption/decryption 902
- Business logic 813
- Business to business 40
- BusyThreads count 899

## C

- CA *See* Certificate Authority
- Cache 807, 854, 865
  - Discards 854
  - Flushing 819
- Cache - Control 531
- Cache appliances 529
- Cache consistency 555
- Cache ID 167, 551, 558, 572, 574
- Cache invalidation 563
  - Event-based 563
  - Policy 563
  - Techniques 563
  - Time-based 563
- Cache monitor 555, 564
- Cache policy 541, 573–574
- Cache replication 543, 555
  - cachespec.xml configuration 562
- Configuration 557
  - Internal messaging server 558
  - Push frequency 561
  - Runtime mode 558
    - Not Shared 561
  - Pull only 560
  - Push and Pull 558
  - Push only 559
- Sharing policy 562
  - Not-shared 562
  - Shared push-pull 562
  - Shared-pull 562
  - Shared-push 562
- Enabling 556
- Testing 563
- Troubleshooting 565

- Cache size 853–854
- CacheMonitor.ear 534
- CacheQueries 581
- cachespec.xml 540, 542, 547, 552, 556, 564
  - Configuration 546, 562
- CacheTimeMargin 581
- Caching 48, 51, 729, 900

- Commands 528
- Content type
  - GIF 528
  - HTML 528
  - JPG 528
  - JSP rendering less dynamic data 528
- Customized content 528
- Database 531
  - DB2 DBCache 531
  - Oracle 9i IAS - Relational Cache 531
  - TimesTen 531
  - Versant enJin 531
- Dynamic content 527–528, 563
- Fragments 531–532
- In memory 571, 584
- Invalidation 527
- Page fragments 530
- Personalized data 527
- Servlets 530, 729
- Static content 527–528, 563, 584
- To disk 584
- Web page design 532
- Web services 530, 729
- WebSphere commands 729

- Caching framework 528
- Caching Proxy 120, 127, 260, 530, 533, 570
  - Cachable content 127
  - Configuration file 261
    - Hostname directive 261
    - Logging directives 262
    - Mapping rules 262
    - Port directive 262
  - Configure 261
  - Configure dynamic caching 581–582
  - Data retrieval 129
  - Dynamic caching 131, 578
    - Cache synchronization 579
  - Forward proxy 127, 129
  - Load balancing 130
  - Monitoring 582
  - Reverse proxy 127
  - Start 261
  - Stop 261
  - Storage device 127
  - Transparent proxy 127
  - Troubleshooting 582
  - Web administration 262
- Caching Proxy adapter module 578
- Caching Proxy statistics 573

- Caching techniques
  - Caching Proxy 584
  - Dynamic caching 584
  - ESI 584
  - For static content 584
  - FRCA 584
  - Web server plug-in 584
- Caching tiers 529
- Cascading resource group 437
- Cascading without fallback 382, 412, 443
- Casting 818
- CBR
  - For HTTP traffic 98
  - For SSL traffic 98
- CBR forwarding 120
- CBR forwarding method
  - Dispatchers component 100, 120
- CBR methods 98
- CBR *See* Content Based Routing
- CDN *See* Content Delivery Network
- Cell 74
  - Master repository 390
  - Synchronization interval 390
- Cell administration 916
- Certificate Authority 143
- Channels 424
- Charts
  - Tivoli Performance Viewer 720
- CheckPoint FireWall-1 440
- CICS 693
- Cipher suite 903
- CIS *See* Customer information service
- Cisco Cache Engine 529
- Cisco CSS Controller 98
- Class
  - Garbage collection 882
  - Instance of 783
  - Structure 818
- Class Instance Statistics 780
- Class Method Statistics 781
- Client authentication 143
- Client ID 617–618
- CLIENT method 644
- CloneID 151, 167
- close method 635
- CLOSE\_WAIT 885
- clruncmd 378, 409, 440
- Cluster 16, 46, 134, 143, 217, 257, 346, 862
  - Backup servers 346, 349
  - Creation 278
  - Primary servers 346, 349
  - Server template 346
- Cluster aware clients 369
- Cluster failover 251
  - IP-based 317
  - Non-IP 317
- Cluster member 16, 139, 143
  - Configuring transports 139
  - Failover operation 196
  - Failure 197
  - Failure during request 204
  - Failure with active sessions 198
  - Marking down 161
  - Overload 207
  - Security 20
  - Stopping 197
- Cluster routing table 232
- Cluster topology definition 381, 411, 442
- Cluster unaware clients 369
- ClusterAddress tag 156, 159
- clverify 383, 413, 443
- cmeckconf 453
- cmmodnet 458
- CMP enterprise beans 856
- CMP *See* Container Managed Persistence
- cmviewcl 452, 456
- Coding best practices 804
- Collection size 806
- com.ibm.CORBA.INTERNAL 248
- com.ibm.CORBA.LocateRequestTimeout 372
- com.ibm.CORBA.MaxOpenConnections 905
- com.ibm.CORBA.NO\_IMPLEMENT 248
- com.ibm.CORBA.RequestRetriesCount 372
- com.ibm.CORBA.RequestRetriesDelay 372
- com.ibm.CORBA.RequestTimeout 372
- com.ibm.CORBA.ServerSocketQueueDepth 904
- com.ibm.ejs.wlm.MaxCommFailures 372
- COMM\_FAILURE 247–248
- COMM\_FAILURE exception 371
- Command Cache 550–551
- Command Design Pattern 551
- Command result cache 583
- Command result caching 550
  - Enabling 551
  - Testing 554
- Commits 909
- Communications failure 248
- COMPLETION\_STATUS 247, 373

- COMPLETED\_MAYBE 247, 373
- COMPLETED\_NO 247, 373
- COMPLETED\_YES 247, 373
- Component failover 321
- Concurrency test
  - Memory leak 875
- Concurrent access resource group 437
- Concurrent requests 696
- Concurrent user 848
- Concurrent Waiters 852
- Conditional check 818
- Config tag 151
- Configuration repository 11
- Configuration synchronization 923
- CONN\_TIMEOUT 470
- Connection
  - Keep-alive 902
- Connection backlog 207, 356
- Connection factory 646
- Connection manager 846
- Connection pool 591, 647, 679, 707, 816, 851–854
  - Maximum connections 600, 602
- Connection pool maximum 610
- Connection pool size 852
- Connection pooling 48, 745, 852
  - Performance data 693
- ConnectionConsumer object 598
- Connections 356
  - Idle 626
- Connection-time failover 486
- ConnectionWaitTimeoutException 470
- ConnectTimeout 207, 890
- Container Managed Persistence 814
- Container transactions 863, 868
  - Bean Managed 868–869
  - Mandatory 868
  - Never 868
  - NotSupported 868–869
  - Required 868–869
  - RequiresNew 868–869
  - Supports 868–869
- Content Based Routing 97, 120
- Content caching tiers
  - Client caching services 529
  - Dynamic caching services 530
  - External caching services 530
  - Internet content delivery services 529
- Content Delivery Network 529, 533
- Continuous availability 323
- Cookies 170, 188, 590, 808, 810
  - JSESSIONID 167
- CORBA 827–828
- CORBA 2.3 specification 220
- CORBA C++ 368
- CORBA CosNaming 220
- corbaloc provider URL 225
- Correlation ID 429, 631
- CosNaming 220
- Cost 60
- Cost/benefit analysis 49
- Count statistic 689, 696
- Counters 692, 696, 707, 714, 717
  - Avg Method RT 691–692
  - Performance data 691
  - Resetting 720
  - Selecting multiple 717
- CPU 588, 842
- CPU utilization 694, 840
- create method 229
- Create server cluster 278
- createQueueSession 595
- CTF *See* Connection-time failover
- Current weight 158
- Customer information service 47
- Customer self-service 38
- CustomLog 890
- CWOF *See* Cascading without fallback

**D**

- Data
  - Caching 818
- Data availability 435
- Data counters 695
- Data high availability 316, 523
- Data integrity 374
- Data server 33
- Data source 816
- Data source queues 852
- Data transmission time 895
- Database
  - Failure
    - Impact on administrative processes 475
    - Impact on applets, servlets, JSPs 477
    - Impact on application servers 476
    - Impact on EJBs 477
    - Impact on Java and C++ applications 476
    - Impact on naming service 477

- Impact on persistent session 476
- Impact on security service 477
- Impact on Web-based clients 477
- Impact on WebSphere WLM 477
- Database connection 817
- Database connection pools 707
- Database indexes 50
- Database instances
  - Single vs multiple 493
- Database manager overhead 909
- Database persistence 168
- Database row lock 867
- Database server 65
  - Remote from application server 65
- Database session management
  - Session persistence 175
- Data-centric application 337
- DB2 65–66, 78, 110, 175, 300, 440, 653, 661, 906
  - Buffpage 908
  - Configuration 299
  - Logging 907
  - MaxAgents 907
  - MaxAppls 907
  - MinCommit 909
  - Multi-row feature 179
  - Query optimization level 908
  - reorgchk 908
  - Sample start script 382, 412, 442
  - Variable row size 179
- DB2 catalog command 301
- DB2 configuration advisor 907
- DB2 DBCache 531
- DB2 driver classes 646, 668
- DB2 Parallel Server 489
- DB2 performance
  - Repeatable read 866
- DB2 transaction log 909
- db2cmd 299
- DB2COMM 907
- db2mscs 467
- db2start 382, 442
- db2stop 382, 443
- Dead Letter Queue 634, 657
- Deadlock 734, 842, 853
- Deadlock Detection Report 741
- Deadlock situation 600
- Deadman switch 448
- Default gateway address 123
- default\_host 137
- DefaultApplication.ear 184
- DeliveryMode.NON\_PERSISTENT 615
- DeliveryMode.PERSISTENT 615
- Demilitarized Zone (DMZ) 63
- Dependency ID 552, 554
- Deployment descriptor 221, 814
- Deployment Manager 6, 10, 78, 390
  - Failure 400
    - Impact on Administrative Console 404
    - Impact on application clients 404
    - Impact on application servers 403
    - Impact on cell naming server 403
    - Impact on cell security server 403
    - Impact on cell WLM runtime service 404
    - Impact on data integrity 401
    - Impact on File Transfer Service 404
    - Impact on Node Agent 402
    - Impact on PMI monitoring 404
    - Impact on RAS Service 404
    - Impact on repository synchronization 401
    - Impact on Synchronization Service 404
    - Impact on wsadmin 404
  - High availability 406
    - plugin-cfg.xml 151
    - SPOF 401, 406
- Deployment Manager high availability without clustering 917
- Deployment Manager node failure 916
- DES 903
- Development cycle 760
- Development environment 136, 153
- Dirty reads 866
- Disaster recovery 323
- Disk array 451
- Disk IO 842
- Disk mirroring 451
- Dispatcher 96
  - Add advisor 107
  - Add cluster 106
  - Add port 106
  - Add servers 106
  - Advisors 96, 110
  - Connecting 266
  - Executor 96
  - Login 267
  - Manager 96
  - Start executor 106
  - Start manager 107
  - Weighted round robin load balancing 107



- Dispatcher component's CBR forwarding method 97
- Distributable 864, 870
- Distributed Fragment Caching and Assembly Support 570
- DNS 110
- DNS redirection 516
- DNS round robin 511
- DNS server 268
- doGet 594
- doGet() 812
- doPost() 812
- Downstream response wait 734
- Downtime 325
- DRS 23, 543
- dsconfig 123
- dscontrol 97, 105, 112, 116
  - executor report 108
  - manager report 107
- dsserver 105, 116, 266
- Durable subscription 618–619
- Durable subscription ID 618
- DynaCacheEsi application 540
- DynaCacheEsi.ear 572
- dynaedge-cfg.xml 580
- Dynamic cache 259, 729
- Dynamic Cache Administrative Console 540
- Dynamic Cache Monitor 534, 540, 573, 729–730
  - Edge Statistics 573
  - Installation 535–539
- Dynamic cache service 528, 533, 695, 900
  - Cache replication 543
  - Configuration 534, 541
  - Disk off-load 541–542
    - LRU algorithm 542
    - Priority weighting 542
  - Enabling 540
- Dynamic caching
  - Cache consistency 555
  - Cache replication 555
  - Command Design Pattern 551
  - WebSphere Command Framework API 551
- Dynamic content 528
- Dynamic SQL statements 908

## E

- Edge of Network Caching Support 583
- Edge server 33, 51

- Edge Side Include *See* ESI
- Edge Statistics 573
- EIS *See* Enterprise Information System
- EJB
  - Deployment descriptor 814
  - InitialContext 813
  - Isolation levels 814
  - Local interface 813
  - Lookup operation 813
  - Object reference 218
  - read-only method 814
  - Reference 813
  - Reference mapping 222
  - Remote method call 813
  - Workload management 215
- EJB 2.0 813, 869
- EJB access beans functionality 814
- EJB caching 374, 815
  - Option A caching 219, 374, 816, 864–865
  - Option B caching 219, 374, 816, 864–865
  - Option C caching 219, 374, 816, 864–865
- EJB client 855
- EJB cluster 283
- EJB container 221, 234, 257, 815, 854
  - Availability 80
  - Cache settings 854
  - Cache size 854
  - Cleanup interval 854
  - Transaction management 814
- EJB container failover 365
  - Behavior 371
  - Tuning 371–372
- EJB container workload management 257
- EJB deployment descriptor 633, 666
- EJB home 222, 813
- EJB reference 221–222
- EJB response time 707
- EJB server selection policy
  - Prefer local 233
  - Server weighted round robin routing 232
- EJB WLM
  - Behavior 239
  - Configuration changes 230
  - Initial request 228
  - Prefer local 233
    - Behavior 243
    - Configuration 237
      - Runtime changes 238
  - Process affinity 230, 233, 240, 243, 246

- Random 234
- Round robin routing 234
- Selection policy 231
- Server weighted round robin
  - Behavior 240
- Server weighted round robin routing 232
  - Configuration 234
  - Runtime changes 236
- Transaction affinity 218, 233, 240, 243, 246
- EJB Workload management 215
- ejbActivate 865
- EJBContext 472
- EJBs 813
  - Caching Option A 219, 374, 864–865
  - Caching Option B 219, 374, 816, 864–865
  - Caching Option C 219, 374, 816, 864–865
- EJS WLM 20
- EJS workload management 14, 341
  - Enabling 216
  - Entity bean 218
  - Failover 246
  - Normal operation 228
  - Server selection policy 232
  - Stateful session bean 218
  - Stateless session bean 217
- Embedded HTTP transport 11, 55, 135, 139
- Embedded JMS 619
- Embedded JMS provider 624
- Embedded JMS server 595, 611, 614–615, 620, 626, 630, 637, 642, 644, 646
- Embedded Messaging (JMS) Server 11
- Embedded WebSphere JMS provider 418
- Enable clone support 617
- Encrypted transport 63
- Enterprise beans
  - Performance data 693
- Enterprise Java Services
  - Workload management 20
- Enterprise JavaBean 855
  - Caching options 26, 219
    - Option A caching 219, 374, 864–865
    - Option B caching 219, 374, 816, 864–865
    - Option C caching 219, 374, 816, 864–865
  - Entity bean 25, 218
  - Entity bean home 217
  - Entity bean instance 217
  - Home 228
  - Passivation 219
  - Selection policy 21
  - Server affinity 27
  - Session 25
  - Session bean home 217
  - Stateful session bean 25, 218
  - Stateless session bean 25, 217
  - Stateless session bean instance 217
  - Transaction 25
  - Workload management 20
- Enterprise JavaBeans 813
- Enterprise Server Manager interface 899
- Entity 814
- Entity bean 25, 218, 852
  - Server affinity 27
- Entity bean access 814
- Entity EJBs 810, 814
  - Activate at 864
  - Load at 864
  - Manipulation of data 814
  - Result sets 814
- Environment
  - Development 136, 153
  - Production 136
- Error condition 818
- error.log 183
- error\_log 183
- ESI 530, 533, 567, 570
  - Best practices 589
  - Performance considerations 588
- ESI cache 572
  - Invalidation 575
- ESI cache statistics 573
  - Cache Hits 574
  - Cache Misses By Cache ID 574
  - Cache Misses By URL 574
  - Cache Time Outs 575
  - Content 575
  - ESI Processes 574
  - Evictions 575
  - Number of Edge Cached Entries 574
- ESI include tags 570, 572
- ESI processor 540, 567, 571, 584
  - Configuration 572
- ESI processor cache 573
- ESI request 574
- ESI surrogate capability 533
- ESI/1.0 530
- esiInvalidationMonitor 540, 572
- EsInvalidator 570
- Event broker

- Execution Group 663
  - Message flow 663
- Exception 818, 842, 909
  - Catching 818
  - NO\_IMPLEMENT 247
  - Throwing 818
  - TRANSIENT 246
- execute() 900
- Execution Flow view 786
- Execution Group 663
- Executor 96
- External cache
  - Invalidation 583
- External Cache Adapter 541
- External caching scenarios 566
  - Configuration
    - Adapter Bean Name 569
    - AFPA 568
    - Caching Proxy 568
    - ESI 572
    - EsilInvalidator 568
    - External cache group 568
      - Cache group members 569
    - Web server plug-in 572
  - IBM HTTP Server's high-speed cache 566
  - Using Caching Proxy 578
    - Configuration
      - External cache group 579
  - Using Edge Components 566
  - Using ESI 566, 570
  - Using FRCA 566, 576
    - Configuration
      - External cache group 576
  - Using Web server plug-in 566, 570

## F

- Facade 246
- Fail back 340, 438
- Fail fast 340
- Fail transparent 342
- Failover 6, 13, 18, 134, 143, 337, 407, 438
  - Active/active configuration 339
  - Active/standby configuration 337
  - Administrative server process 475
  - Adoptive server 364
  - IP-based database 475
  - Moving cluster resources 438
  - Non-IP-based database 492

- ORB plug-in 246
- Primary and backup servers 163
- Programming transparent 468
- Session persistence 173
- Terms and mechanisms 337
- Transaction log 376
- Tuning 207
- Web server plug-in 161, 163
- Failover data service 318
  - Active/Active mode 318
  - Active/Standby mode 318
  - Hot standby 318
- Failover time 336
- Failover tuning
  - ConnectTimeout 207
  - MaxConnections 210
  - RetryInterval 210
- Failover unit 377–378, 382, 412
  - Access points 377
  - Applications 377
  - Dependencies 377
- Failure
  - Deployment Manager 400
  - JMS server 420
  - Node Agent 391
- Fallback 340, 438
- Fast Response Cache Accelerator *See* FRCA
- Fault isolation 8, 374
- Fault-detection time 336
- Field
  - Static 808
- filemon 883
- Filter 728
- FIN\_WAIT\_2 885
- Final modifier 818
- finally block 635
- findByPrimaryKey 867
- Firewall 9, 63
  - High availability 514
- Five nines 323
- Floating IP address 458
- Fragment caching 531–532
- Fragmentation 805
- FRCA 533, 566, 568–569, 576–577
  - Monitoring cache 578
- FTP 110
- Function-centric application 337

## G

- Garbage collection 701–702, 806, 817, 849, 877
  - Algorithm 880
  - Bottleneck 873
  - Class 882
  - Common problems 879
  - Compaction phase 872
  - Concurrent mark mode 872
  - Generation 880
  - HotSpot JVM 880
  - Incremental compaction mode 873
  - Mark phase 871
  - Monitoring 873
  - Parallel mark mode 872
  - Parallel sweep mode 872
  - Phases 871
  - Sweep phase 871
  - Time 808
- Garbage collection bottleneck 874
- Garbage collection call 808
  - Duration of 877
  - Length of 877
  - Number of 877
  - Time between calls 877
- Garbage collection calls 874
- Garbage Collector 805, 871
  - Mark - Sweep - Compact technique 871
- GC See Garbage collection
- Generation garbage collection 880
- Generic JMS call 608
- Generic JVM arguments 702
- genericJvmArguments 705
- GenPluginCfg
  - wsadmin 153, 155
- GenPluginCfg command 153, 155
- get-by-correlationId 631
- get-by-messageId 631
- getRuntimeEnvInfo method 229
- getter type method 814
- Global Security 711
- GNU 827
- GNU General Public License 827
- godm 378, 409, 440
- Graphical profiling view 761
  - Execution flow 761
  - Method Invocations 761
- Grinder 835

## H

- HA hardware 643
- HA software 616, 643
- hacf 465
- HACMP 378, 407–409, 421, 436, 440, 616, 653
  - Application server 382, 412, 442
  - Cluster verification 383, 412, 443
  - Cycles to failure 448
  - DB2 sample stop script 382, 412, 443
  - Failure detection rate 448
  - Failure tests 446
  - Heartbeat rate 448
  - Oracle sample start script 443
  - Oracle sample stop script 443
  - Resource group 382, 412
  - Takeover verification 383, 413, 444
  - Tuning 447
  - Typical failover 383, 413, 444
- HACMP 4.5 436
- HACMP configuration
  - WebSphere Deployment Manager Failover Unit 412
  - WebSphere Network Deployment part 410
- HACMP/ES 436
  - Application server 382, 412, 442
- Handshake 902
- Hang condition 734
- Harden Get Backout 639
- Hardened 639
- Hardware
  - Requirements 256
- Hardware accelerator 903
- Hardware-based high availability 6
- hareg 462
- Hashtable 874, 876
- hastat 463
- haswitch 463
- Heap 626, 702, 843, 863
  - Allocations 701
  - Compaction 808
  - Consumption 876
  - Fragmentation 876
  - Number of objects in heap 808
  - Parameters 871, 877
  - Size 745
  - Space 51
  - Xcompactgc 876
  - Xgcpolicy 872
  - Xmaxe 872

- Xmaxf 872
- Xmine 872
- Xminf 872
- Xms 872
- Xmx 872
- Heap compaction 872
- Heap expansion 872
- Heap shrinkage 872
- Heap size 808, 874, 881
  - Initial 877, 881
  - Maximum 873, 877, 881
  - Minimum 873
  - Tuning 878
- Heap thrashing 878
- Heartbeat 101, 444
- Heavy load conditions 900
- High availability
  - Adoptive server 364
  - Basic systems 321
  - Best possible topologies 333
  - Business operation hours and pattern 325
  - Causes of downtime 325
  - Clustering 316
  - Component failover 321
  - Continuous 323
  - Data 316, 523
    - Types 524
  - Data management 435
  - Database failures 317
  - Data-centric application 337
  - DB2 database server 317
  - Deployment Manager 317, 406
    - OS daemon 406
    - Using clustering software 406
  - Disaster recovery 323
  - EJB bootstrap failover 366
  - EJB client redundancy 366
  - EJB container failover 365
  - EJB container redundancy 367
  - Embedded JMS server
    - Using clustering software 421
  - Fail back 340, 438
  - Fail fast 340
  - Fail transparent 342
  - Failover 337
  - Failover data service 318
  - Failover unit 377
  - Fallback 340, 438
  - Firewall 317, 514
    - Using clustering software 515
    - Using Interactive Session Support 517
    - Using Load Balancer 518
    - Using network sprayer 516
  - Five nines 323
  - Function-centric application 337
  - HACMP
    - Advantages 387
  - HACMP/ES 317
  - Horizontal scaling 322
  - Hot replacement 319
  - Hot standby 318
  - HP MC/ServiceGuard 317
  - HTTP server 317
  - HTTP session failover 357
    - Ordered routing process 359
    - Server ID 357
    - Session ID 358
  - IBM HACMP 317
  - IP address takeover 437
  - IP-based cluster failover 317
  - LDAP 507
  - LDAP server 317
  - Levels 319
  - Load Balancer 504
    - Backup server 506
    - Primary server 506
    - Standby mode 506
  - Location Service Daemon failover 371
  - Loopback alias configuration 398
  - MDBs 417
  - Messaging 417
  - Mutual takeover configuration 438
  - Network Dispatcher 504
  - Network failures 354
  - Network file system 521
    - Using clustering software 521
  - Node Agent 317, 398
    - OS daemon 399
    - OS daemon configuration 398
    - Using clustering software 399
  - Non-IP cluster failover 317
    - Oracle Parallel Server 318
  - Operating system TCP time-out value 353
  - Oracle Parallel Server 317
  - OS-clustering 431
  - Parallel database server 317
  - Performance availability requirement 325
  - Process 316, 523

- Real Application Cluster
- Redundant data 321
- Scalable data service 319
- SPOF 326
- Sun Cluster 317
- System failover 322
- System uptime percentage 325
- Techniques
  - Transparent Application Failover
- Transaction log failover 376
- Two-level failover 349
- Two-phase transaction 376
- Uncoordinated processes 340
- Unit of failover 317
- Vertical scaling 322
- Web server 498
- WebSphere availability levels 328
  - HA level 1 328
  - HA level 2 329
  - HA level 3 330
  - HA level 4 330
  - HA level 5 331
  - HA level 6 332
- WebSphere MQ
  - Queue Manager cluster 425
  - Using clustering software 424
- WebSphere MQ cluster and HACMP 430
- WebSphere MQ with Load Balancer 429
- WebSphere process failures 317
- HitCount 185, 190
- Hits per
  - Day 822
  - Hour 822
  - Minute 822
  - Month 822
- Home 217, 228
- Horizontal scaling 19, 70
- Host alias 535
- Hosts file 268
- Hot replacement 319
- Hot standby 318
- HotSpot JVM *See* Sun JDK 1.3 HotSpot
- HP 449
- HP UX 450
- HP-UX 11i 884
- HP-UX 11i tuning
  - JVM virtual page size 884
  - TCP\_CONN\_REQUEST\_MAX setting 884
- htadm 128
- htcformat 582
- HTML frames 809
- HTTP 110, 135
- HTTP cache 531
- HTTP cache directives 529
- HTTP GET 824, 831
- HTTP POST 824
- HTTP requests 819
- HTTP response code 503 212
- HTTP server *See* Web server
- HTTP session 22, 808
  - Failover 357
  - Server affinity 27
  - Timeout 9
- HTTP session failover 357–359
- HTTP session memory-to-memory replication 543
- HTTP transport 11, 138, 140–141, 861
  - Custom properties 860
  - Embedded 55, 135
- HTTP transport port 535
- HTTP/1.1 530–531
- HTTP/1.1 caching directives 531
- http\_plugin.log 180, 726
- httpd command 896
- HTTPD process 894
- httpd.conf 577, 730, 732, 890–891, 893, 895
- HTTPS 135
- HTTPS transport 140–141
- HttpSessionBindingListener 810

**I**

- IBM 7133 410
- IBM Agent Controller 762
- IBM AIX 5.2 256
- IBM alphaWorks 793
- IBM DB2 UDB Client version 8.1 644
- IBM DB2 UDB V8.1 256
- IBM Edge Components 573
- IBM HTTP Server 10, 78, 110, 530, 533, 583, 731, 862, 888, 892
  - Error log 183
  - Monitoring 730
  - Select performance data 731
  - Server-status page 731
  - Use as external cache 576
- IBM HTTP Server 2.0.42.2 256
- IBM HTTP Server powered by Apache 824
- IBM Java 2 SDK version 1.4.1 878

- IBM JDK 1.4.1 256
- IBM ORB 228
- IBM Rational Suite TestStudio 835
- IBM service log file 733
- IBM Tivoli Monitoring for Transaction Performance 722
- IBM Tivoli Monitoring for Web Infrastructure 722
- IBM Tivoli performance monitoring tools 705
- IBM Tivoli Storage Manager 915
- IBM WebSphere Application Server Network Deployment V5.1 256
- IBM WebSphere Edge Components V5.1 256
- IBM WebSphere Studio Application Developer 759, 804
- IBM WebSphere Studio Application Developer V5.0 788
- ibmproxy.conf 127, 581–582
- ibm-web-ext.xml 152
- ID
  - hexadecimalnumber 632
- Identifying field 632
- ifconfig 104, 123, 266
- if-then 818
- IHS 1.3.28 894
- IHS 2.0 894
  - Directives 896
  - mod\_deflate 895
  - Multi-processing 896
  - Single-processing 896
  - Tuning
    - MaxClients 897
    - MaxRequestsPerChild 897
    - MaxSpareThreads 898
    - MinSpareThreads 898
    - ServerLimit 898
    - StartServers 898
    - ThreadLimit 897
    - ThreadsPerChild 897
- IHS *See* IBM HTTP Server
- IHS2.0 576
- IIO requests 368
- IIO *See* Internet Inter-Orb Protocol
- IIS permission properties 899
- Impact rating 696
- IMS 693
- InfoLibria DynaCache 529
- Inheritance 814
- init() 870
- Initial heap size 877
- InitialContext 220, 225, 372, 813
- Initialization
  - Lazy 806
- initialization routines 870
- In-memory cache 571
- In-process request 246
- Installation
  - Using response files 920
- Instance 217, 691
- Instance Statistics 783
- Instrumentation level 692, 696, 700–701, 712–713, 717
  - For enterprise bean methods 715
  - Set 700
  - Set using Administrative Console 700
  - Set using wsadmin 701
  - Setting 696
- Instrumentation level setting
  - Custom 700
  - None 700
  - Standard 700
- Interactive Session Support 517
- Internet Information Server 10
- Internet Inter-Orb Protocol 64
- Internet Services Provider 529
- Interoperable Object Reference 229
- Invalidation 561, 563
  - ESI cache 575
  - Event-based 563
  - Event-driven 566
  - External cache 583
  - Policy 563
  - Techniques 563
  - Time-based 563
- Invalidation API 563
- IOR
  - Direct 229
  - Indirect 229
- IOR *See* Interoperable Object Reference
- iostat 910
- IP address takeover 437
- IP sprayer *See* Load Balancer
- IP-forwarding 129
- iPlanet 898
- iSeries 709
- Isolation levels 814, 863, 865
  - database overhead 866
  - row locking 866
- ISP *See* Internet Services Providers

ITSO sample topology 78, 256

## J

J2C connectors

Performance data 693

J2EE 54, 215

Application 901

Business logic tier 55

Client tier 54

Components 134

Enterprise Information System tier 55

Multi-tier environment 54

Presentation tier 55

Request Profiler 762

Request Profiler Agent 786

Specification 62, 812

Static Web content 62

Tiers model 54

J2EE 1.3 813

Messaging 418

J2EE 1.3 specification

JMS 423

J2EE application client container 224

J2EE applications 591

J2EE client application

Prefer local policy 234

J2EE Connector Architecture 693

J2EE management specifications 689

J2EE name 618

Java

Access to synchronized resources 810

Casting 818

Class structure 818

Conditional check 818

Copying contents of arrays 818

Final modifier 818

if-then 818

Monitors 810

Reflection 817

Serialization 809

Strings 817

Synchronization 810

Using ? for conditional check 818

Java 2 Platform Enterprise Edition 54, 215

Java Buffered I/O classes 818

Java client 13, 217, 233

Java Database Connectivity *See* JDBC

Java I/O libraries 811

Java Management Extension 688

Java Management Extension (JMX) interface 723

Java memory management 806

Java memory tuning 870

Java Message Service (JMS) Server 11, 418, 808

Configuration 301

Java Messaging Service *See* JMS

Java Naming and Directory Interface *See* JNDI

Java performance

Swapping 881

Java process

Attach to 767

Java process ID 239

Java profiler 874

Java Profiling Agent 762

Java Transaction API 694

Java version 1.4.1 28

Java Virtual Machine 56, 246, 693

Java Virtual Machine API 693

Java Virtual Machine Profiler Interface 762

java.lang.String 817

java.lang.StringBuffer 817

java.rmi.RemoteException 868

java.sql.Connection 817

java.sql.PreparedStatement 817

java.sql.ResultSet 817

java.sql.Statement 817

java.util.ArrayList 811

java.util.Hashtable 806, 811

java.util.Vector 806, 811

java: lookup names 226

javax.jts.TransactionRequiredException 868

javax.servlet.Servlet.init() 812

javax.transaction.UserTransaction 868

JCA 808

JCA connection pools 681

Monitors 681

JDBC 635, 808, 814, 816

Driver 817

Resources 301

JDBC calls

Direct 469

Indirect 474

JDBC Provider 176, 852

Connection pool settings 852

JDBC resources 644, 668

DB2 driver classes 646

JFS log 379, 410, 441

JFS *See* Journaled File System



- JIT compiler *See* Just in Time compiler
- JMeter 805, 835
- JMS 556, 591, 593
  - Average workload arrival rate 623
  - Client subscription
    - Durable 617
    - Non-durable 617
  - Client-side selection 631
  - Components 591
    - JNDI Name service 593
    - Listener port 596
    - Message queue 593
    - Queue connection factory 595
    - Queue destination 593
    - Queue manager 593, 595
    - Target queue 595
  - Configuration guidelines 610
  - Correlation ID 631
  - Durable subscription 619
  - Durable subscription ID 618
  - Embedded 591
  - Failure 628
  - Message filtering by content 630
  - Message ID 631
  - Message persistence 635
  - Message queue 593
  - Peak workload arrival rate 622
  - Performance tuning 623
  - Point-to-point 608
  - Publish and subscribe 608
  - Publish/subscribe 617
  - Queue destination settings 615
  - Queue reader thread 623
  - Server-side selection 631
  - Topic destination settings 615
- JMS API 593
  - Receive messages 593
  - Send messages 593
- JMS class 615
  - QueueSender 615
  - TopicPublisher 615
- JMS component settings 614
- JMS components 605
- JMS configuration
  - Client ID 617
  - Enable clone support 617
  - Enable XA support 616
  - Maximum messages 624–625
  - Maximum session pool size 628
  - Maximum sessions 621–623
  - Message persistence 614
  - Transport Type
    - BINDINGS 614
    - CLIENT 614
    - DIRECT 620
- JMS Connection Factories 682
- JMS infrastructure 605
- JMS objects 618, 635
  - Use of 609
- JMS provider 598, 631
  - Configure Connection Factories 419
  - Configure MDB listener ports 420
  - Embedded 418
  - Generic 418
  - WebSphere MQ 418
- JMS provider code 638
- JMS queue 665
- JMS resources 644, 649
- JMS *See* Java Message Service (JMS) Server
- JMS server
  - CLIENT method 644
  - Embedded 595
  - Failure 420, 644
  - HA approaches 421
  - SPOF 421
- JMS specification 631
- JMS statistics 680
- JMS usage 593
- JMX 390, 687–688
- JNDI 228, 816, 818
- JNDI lookup 221, 595, 598, 646
  - Server cluster 227
  - Single server 226
- JNDI name 612
  - Fully qualified 221
- JNDI name resolution 221
- JNDI Name service 593
- JNDI name service 595
- JNDI namespace 594
- Journalized File System 883
- JSESSIONID 167, 170, 547
- jsp include tag 811
- jsp usebean tag 811
- JTA 694
- Just In Time compiler 881
- JVM 56, 233, 530, 623
- JVM log 564
- JVM mode

- client 880
- server 880
- JVM resource 626, 635
- JVM tuning 880
- JVMPI 687, 693, 701, 762, 841, 873
  - Disabling using Administrative Console 704
  - Disabling using wsadmin 705
  - Enabling using Administrative Console 702
  - Enabling using wsadmin 703
- JVMPI profiler 874
- JVMPI *See* Java Virtual Machine Profiler Interface

## K

- keep-alive 902
- keep-alive connection 902

## L

- lanscan 457
- Large remote object size 734
- Latency 35, 901
- Lazy initialization 806
- lbadmin 105, 266
- LDAP 9, 901
- Least recently used *See* LRU algorithm
- Life cycle 693, 805
- Lightweight Directory Access Protocol
  - High availability 507
- Linux scheduler 885
- ListenBackLog 900
- Listener 607
- Listener port 596, 618, 625, 646
  - Failure 636
  - Initialisation 597
  - Maximum number of sessions 621
  - Maximum retries 638
  - Maximum sessions 602
  - Retry interval 638
  - Retry settings 636
- Listener port configuration 598
- Listener port sessions 598, 607, 624
  - Maximum number of 605
- Listener service thread pool 626
- Load at 864
  - Activation 864
  - Transaction 864
- Load Balancer 83, 96
  - Active cookie affinity 119–120
  - Add cluster 106

- Administration GUI 266
- Advisor logging level 117
- Advisor status 116
- Backup server 506
- CBR component 120
- Cisco CSS Controller 98
- Cluster address 101, 265, 508
- Collocated servers 99
- Commands
  - dsserver 266
  - lbadmin 266
- Configuration 264
  - Add cluster 269
  - Add port 270
  - Add Web servers to cluster 271
  - Start Executor 268
- Content Based Routing 97
- Custom advisors 83, 112
- Dispatcher component 96
- Enhance WebSphere MQ scalability 429
- High availability 101, 504
- Installation and Configuration 103
- Metric server 97
- Mutual High availability 102
- Nortel Alteon Controller 98
- On dedicated server 99
- Passive cookie affinity 119
- Primary server 506
- Script
  - goActive 506
  - goInOp 506
  - goStandby 506
- Server affinity 118
- Site Selector 98
- SPOF 101
- SSL session ID 120
- SSL session ID affinity 119
- Start 105, 266
- Start/Stop 266
- Sticky time 119
- Stickyness to source IP address 119
- Stop 266
- Topologies 99
- URI affinity 119–120
- Web server cluster 104
- Load balancing 6, 13, 17
  - Method 153
- Load information 694
- Load on startup 864, 870

- Load testing tools 823
- Load value 693
- LoadRunner 439, 450, 835
- Local cache 530
- Location Service Daemon 229, 368–369, 390
  - Failover 371
- Log
  - System.out 725
  - Tivoli Performance Viewer 719
  - Viewing with Tivoli Performance Viewer 707
- Log Analyzer 733
  - Symptom database 733
- Logging 812
  - Reduction of 812
- Logging library
  - Multithreaded 812
- Logical host 317
- LogLevel 180
- Logout functionality 809
- Long-running test
  - Memory leak 874
- Lookup 816
- Loop 818
- Loop iteration 818
- Loopback alias 265
  - Configuration 398
- Lotus Domino 10
- Low load reaching application server 734
- LRU algorithm 542
- LSD *See* Location Service Daemon
- Isdev 379, 409, 441
- Isnrctl 443, 485
- Ivcreate 452
- Ivextend 452

## M

- mac forwarding 100
- magnus.conf 899
- Maintainability 4, 7, 60, 63, 69, 71
  - Configuration
    - Dynamic 8
    - Mixed 8
  - Fault isolation 8
- Managing state 21
- Mandatory 814, 868
- Master configuration repository 916
- Master repository 390
- Max Application Concurrency 850

- Max Connections 852–853
- MAX.RECOVERY.RETRIES 636
- MaxClients 890–891, 897
- MaxConnections 151, 210
- Maximum concurrency level 851
- Maximum concurrency point 850
- Maximum heap size 873, 877
- Maximum messages 624–625
- Maximum number of messages 623–624
- Maximum number of threads 207, 897
- Maximum sessions 621–623, 646
- Maximum size 856
- Maximum thread pool size 629
- Maximum time to recover 319
- Maximum weight 158
- MaxKeepAliveConnections 858–859
- MaxKeepAliveRequests 859–860
- MaxRequestsPerChild 892, 897
- MaxSpareServers 891
- MaxSpareThreads 898
- maxWeight 158
- MBeans 689
- MC/ServiceGuard 407, 421, 448
  - Cluster verification 454
  - Daemon processes 450
  - DB2 sample service start script 453
  - DB2 service stop script 453
  - Failover verification 454
  - Failure tests 456
  - Heartbeat interval 459
  - Network polling interval 459
  - Node time-out 459
  - Oracle sample service start script 454
  - Oracle service stop script 454
  - Resource polling interval 459
  - Tuning 458
  - Typical failover 454
- MDB 593, 596, 598, 605, 618, 638, 640
  - onMessage 622, 624
- MDB listener port 620
- MDB listener ports 619
- MDBs 26, 428, 816, 818
- Mean time between failures 319
- Memory 693, 842
  - Utilization 873
- Memory allocation error 884
- Memory allocation fault 871
- Memory leak 805, 807, 817, 838, 871, 873–874, 877

- Concurrency test 875
- Long-running test 874
- Repetitive test 875
- System instability 874
- System test 875
- Testing 874, 876
- Tivoly Performance Viewer 876
- Memory overuse 874
- Memory-to-memory replication 23, 59, 168, 347, 809
  - Client-server 178, 365
  - Peer-peer 178, 365
  - Replication domain 179
  - Replicator entry 179
- Mercury LoadRunner 356, 439, 450, 775, 805, 823
- Message backlog 622
- Message broker 618, 638, 661
  - Connection timeout 638
- Message Broker version 5 621
- Message Driven Beans 26, 596, 816, 818
- Message ID 631
- Message listener
  - Service thread 604
- Message listener service 596–597, 599, 605, 633, 636
  - Maximum sessions 625
  - Maximum threads 602
- Message listener threads 607, 624, 679
- Message persistence 614, 635, 649
  - Configuration values 615
  - NON\_PERSISTENT 649
- Message queue 593, 651
- Message retention
  - Disable 634
- Message selector 633
- Message-Driven Beans 593
- Message-oriented Middleware 422
- Messages
  - Point-to-point 605, 657
- Messaging
  - Load balancing 425
  - Non-persistent 620
  - Non-transactional 620
  - Point-to-point 418
  - Publish/subscribe 418
- Messaging system 593, 611
  - Topology 612
- Messaging system topology 623
- Messaging workload 643
- Method extensions 865
- Method Statistics 789
- Microsoft Cluster Service 407, 421, 465
  - Cluster configuration 466
  - Microsoft Windows NT Server Enterprise Edition 465
  - Windows 2000 Advanced Server 465
- Microsoft Data Access Components 828
- Microsoft IIS *See* Microsoft Internet Information Server
- Microsoft Internet Information Server 10, 899
- Microsoft Windows 2000 Server 256
- Min connections 853
- MinCommit 909
- Minimum heap size 873
- Minor code 248
- MinSpareServers 891
- MinSpareThreads 898
- Mirrored backup cluster 249
- mod\_deflate 895
- mod\_mem\_cache 584
- Model, View, and Controller 551
- Module
  - Performance data 690
- MOM *See* Message-oriented Middleware
- MPM architecture 896
- mpm\_winnt 896–897
- mpm\_worker 896
- MQ
  - Transaction support 614
- MQ cluster 428
- MQ header 631
- MQ JMS classes 619, 626
- MQ JMS connection pooling 626
- MQ JMS pooling 626
- MQ JMS provider 619
- MQJMS.POOLING.THRESHOLD 626
- MQJMS.POOLING.TIMEOUT 626
- mqsisstart 665
- MS Loopback Adapter 265
- MSCS *See* Microsoft Cluster Service
- MSGRETENTION(YES) 634
- MTBF 319
- MTTR 319
- Multi-Processing Modules architecture 896
- Multithreaded logging library 812
- Multi-tier environment 54
- Mutual failover 318
- MVC 551, 589

## N

- Name space 220
  - Cell 220
  - Federation 220
- NAT *See* Network Address Translation
- native\_stderr.log 878
- ndadmin 97
- NDAdvisor.java.servlet 113
- ndcontrol 97
- netasst 485
- netstat 104, 858–859, 884
- Network Address Translation 64, 100
- Network Appliance 529
- Network bandwidth 895
- Network Deployment Manager 6, 10
- Network Dispatcher
  - Cluster address 104
  - High availability 504
  - WebSphere custom advisor
    - Sample 113
- Network Dispatcher manager
  - Start 273
- Network failures 354
- Network file system
  - High availability 521
- Network router address 123
- Network utilization 842
- Never 868
- NFA *See* Non-forwarding address
- nleaf 909
- nlevels 909
- no command 889
- No load reaching application server 734
- NO\_IMPLEMENT exception 247
- NO\_RESPONSE 247–248
- Node
  - Performance data 690
- Node Agent 10, 390, 714
  - Failure 391
    - Impact on Administrative Console 397
    - Impact on application clients 396
    - Impact on application servers 392
    - Impact on Deployment Manager 394
    - Impact on File Transfer Service 397
    - Impact on LSD 394–395
    - Impact on naming servers 395
    - Impact on PMI and monitoring 397
    - Impact on RAS service 397
    - Impact on Security server 396

- Impact on Synchronization Service 397
  - Impact on wsadmin 397
  - High availability 398
- Node failure 916
- Non-blocking connection 208, 347
- Non-durable subscription 617
- Non-forwarding address 102, 123
- Non-IBM ORB 368
- Nonrepeatable reads 866
- Non-serializable data 558
- Nortel Alteon Controller 98
- NotSupported 814, 868–869
- nsswitch.conf 462
- numRequest 696

## O

- Object
  - Collection 807
  - Creation 806
  - Destruction 806
  - Loitering 807
  - Pool 806
  - Reference 805, 807
  - Reuse 806
  - Serializing 809
  - Temporary 807
- Object pools 695, 838
- Object References Table 784
- Object Request Broker 216, 229, 693–694, 846, 855–857, 903
  - Connection cache maximum 905
- On demand computing 56
- Online banking 35
- Online shopping 35, 37
- Online trading 35, 39
- onMessage 596
- onMessage method 599, 622, 624, 679
- Open System Testing Architecture *See* OpenSTA
- OpenLoad 835
- OpenSTA 805, 827
- Operations ratio 847
- OPFS 460
- OPS 317
- OPS/RAC
  - METHOD
    - BASIC 487
    - PRECONNECT 487
- Optimistic read 815

- Optimistic update 815
- Optimization level 908
- Option A caching 219, 374, 816, 864–865
- Option B caching 219, 374, 816, 864–865
- Option C caching 219, 374, 816, 864–865
- Oracle 65, 440
- Oracle 8i 436
- Oracle 9i 436
- Oracle 9i IAS - Relational Cache 531
- Oracle Parallel Server 317, 440, 460, 478
- Oracle performance
  - Read committed 866
- Oracle Real Application Cluster 440
- ORB 216, 693–694, 846, 855–857, 903
- ORB plug-in 216, 246
  - Failover 246
  - Normal operation 228
  - Workload management 341
- ORB Service 856
- ORB thread pool size 855, 905
- org.omg.CORBA.COMM\_FAILURE 373
- org.omg.CORBA.COMM\_FAILURE exception 375
- org.omg.CORBA.NO\_IMPLEMENT 373
- org.omg.CORBA.NO\_RESPONSE 373
- org.omg.CORBA.NO\_RESPONSE exception 375
- org.omg.CORBA.TRANSIENT 248
- OS TCP/IP keep-alive time-out 372
- OSE Web server plug-in 65
- Out of Memory exception 874
- Over-utilizing objects 874, 877

## P

- Package 317
- Package IP address 458
- Package statistics 779
- Page Detailer 542, 759, 792
  - Connection Attempt Failed 794
  - Connection Setup Time 794
  - Considerations 797
  - Data capture 794
  - Delivery Time 795
  - Details view 800
  - Detect broken links 798
  - Detect server timeouts 798
  - Host Name Resolution 794
  - Legend 799
  - Measure Web application performance 792
  - Monitoring HTTP and HTTPS requests 793

- Page Time 794
- Performance measurement
  - Browser cache 797
  - Network delays 797
  - Packet loss 797
  - Server Response Time 794
  - Socks Connection Time 794
  - SSL Connection Setup Time 794
- Paging activity 842
- Partitioning key 491
- Pass by reference 814, 903
- Pass by value 246, 814, 903
- Passivation 219
- Passive cookie 119
- Peak load 839
- Performance tuning
  - DB2 log files 883
  - Web server reload interval 888
- Percent Maxed 855, 857
- Percent Used 852
- Perf MBean 706
- PerfMBean 689
- Performance 5, 62, 66, 69, 94, 528
  - Access intent read 867
  - Breaking point 822
  - Caching of data 818
  - Caching of EJB references 813
  - Connection pool 816
  - Data counters 695
  - EJB home 813
  - EJBs 813
  - Garbage collection 871
  - Hardware capacity 845
  - Hits per day 822
  - Hits per hour 822
  - Hits per minute 822
  - Hits per month 822
  - Inefficient settings 746
  - Load 728
  - Load testing tools 823
    - ApacheBench 824
      - Advantages 825
      - Limitations 824
      - Options 826
    - Mercury LoadRunner 823
    - OpenSTA 827
      - Architecture 828
      - Collectors 832
      - Commander 829

- Create test 832
- Define test 831
- Execute test 833
- Features 827
- Name Server 829
- Prerequisites 828
- Randomization of requests 831
- Record a script 829
- Repository 829
- Repository host 829
- Script Modeler 829
- Specify runtime parameters 832
- Task group 831
- View test results 833
- Rational TestStudio 823
- Segue SilkPerformer 823
- Maximum number of users 822
- Measuring 760
- Minimize memory usage 806
- Monitoring 707
- Monitoring - tuning - testing 841
- Monitoring tools 687
- Overhead 810
- Peak interval 822
- Peak request rate 822
- Poor coding practices 823
- Profiling 760
- Project cycle 760
- Recommendations 746
- Reduce EJB overhead 813
- Strings 817
- Testing tools
  - Grinder 835
  - IBM Rational Suite TestStudio 835
  - JMeter 835
  - LoadRunner 835
  - OpenLoad 835
  - Segue SilkPerformer 835
  - TestMaker 835
  - TestNetwork 835
  - WebLOAD 835
  - WebStone 835
- Total concurrent users 822
- Trade3 259
- Trade3.1 298
- Traffic patterns 822
- Tuning 821
  - Top-ten monitoring list 836
- Tuning parameter hotlist 844
- Tuning values 821
- User base 822
- Vertical scaling 808
- Web server 73
- Web site performance improvement 798
- Performance Advisor 841
- Performance Advisor in Tivoli Performance Viewer
- See TPV Advisor
- Performance Advisors 745, 836
  - Runtime Performance Advisor 746
  - TPV Advisor 746
- Performance analysis 838
  - Load test
    - Measure steady-state 842
    - Ramp-down time 842
    - Ramp-up time 842
  - Production level workload 839
  - Repeatable tests 840
  - Saturation point 840
  - Stress test tool 839
  - Terminology
    - Load 839
    - Peak load 839
    - Requests/second 839
    - Response time 839
    - Throughput 839
- Performance data
  - Connection pooling 693
  - Counters 691
  - Enterprise beans 693
  - J2C connectors 693
  - Module 690
  - Node 690
  - Server 690
  - Submodule 690
- Performance data classification
  - Boundary statistic 690
  - Bounded range statistic 690
  - Count statistic 689
  - Group 690
  - Load 690
  - Numeric 690
  - Range statistic 690
  - Statistical 690
  - Time statistic 690
- Performance Data Framework 688
- Performance data hierarchy 690
- Performance improvements in V5.1 28
- Performance monitoring 705

- Develop monitoring application 723
- Runtime behavior 760
- Under load conditions 686
- Performance monitoring and management tools 722
- Performance Monitoring Infrastructure API 687
- Performance Monitoring Infrastructure 687, 701, 705
- Performance Monitoring Infrastructure *See* PMI
- Performance Monitoring Service 645, 651, 679
- Performance monitors 680
- Performance of a Web page
  - Key factors 798
- Performance problems 686
  - Application design 686
  - Application view 686
  - Backend system 686
  - End-user view 686
  - External view 686
  - Hardware 686
  - Monitoring tools 686
  - Network 686
  - Product bugs 686
  - Response time 686
- Performance Servlet 687
- Performance testing 822, 839
- Performance tuning 66, 805
  - Access log 890
  - AIX 883
  - AIX file descriptors 883
  - AIX ulimit 883
  - AIX with DB2 883
  - DB2 906
  - DB2 Buffpage 908
  - DB2 configuration advisor 907
  - DB2 logging 907
  - DB2 MaxAgents 907
  - DB2 MaxAppls 907
  - DB2 MinCommit 909
  - DB2 on Linux 907
  - DB2 query optimization level 908
  - DB2 reorgchk 908
  - Disabling security 901
  - Disk speed 845
  - Dynamic cache service 900
  - Full duplex 846
  - HTTP transport custom properties 860
    - ConnectionIOTimeout 860
    - ConnectionKeepAliveTimeout 861
    - ConnectionResponseTimeout 860
    - KeepAliveEnabled 861
    - MaxConnectBacklog 860
    - MaxKeepAliveConnections 861
    - MaxKeepAliveRequests 861
    - Number of concurrent connections 860
  - IBM HTTP Server Linux 892
  - IBM HTTP Server WebSphere plug-in 888
  - IBM HTTP Server Windows 892
    - ThreadsPerChild 892
  - Java memory 870
  - Linux IBM HTTP Server 892
  - Logging 890
  - Max Semaphore 885
  - MaxClients 890
  - MaxSpareServers 891
  - Microsoft IIS 899
  - Microsoft IIS expected hits per day 900
  - Microsoft IIS ListenBackLog 900
  - Microsoft IIS memory allocation 900
  - MinSpareServers 891
  - Network 846
  - Number of requests 847
  - Ongoing process 823
  - Operating System 883
  - ORB 903
  - OS 883
  - Pass by reference 903
  - Pass by value 903
  - Process
    - Testing - Evaluating - Tuning 823
  - Processing time 847
  - Processor speed 845
  - RetryInterval 888
  - Security 901
  - Solaris 885–886
  - Solaris file descriptors 885
  - Solaris kernel semsys
    - seminfo semume 885
  - Solaris tcp\_conn\_req\_max\_q 886
  - Solaris TCP\_FIN\_WAIT\_2\_FLUSH\_INTERVAL 885
  - Solaris TCP\_KEEPALIVE\_INTERVAL 885
  - Solaris TCP\_TIME\_WAIT\_INTERVAL 885
  - Solaris tcp\_comm\_hash\_size 886
  - Solaris tcp\_xmit\_hiwat 886
  - Solaris ulimit 885
  - Solaris virtual page size 886
  - SSL 902



- SSL hardware accelerator 903
- StartServers 891
- Sun JDK 1.3 HotSpot 880
- Sun ONE Web Server 898
- Sun ONE Web Server Active Threads 898
- System memory 845
- ThreadsPerChild 891
- Top-ten monitoring list
  - Average response Time 836
  - CPU utilization 837
  - Datasource connection pool size 836
  - Disk and network I/O 837
  - EJB container thread pool 836
  - Garbage collection statistics 837
  - JVM memory 837
  - Live number of HTTP Sessions 836
  - Number of requests per second 836
  - Paging activity 837
  - Web container tread pool 836
  - Web server threads 836
- Transaction log 906
- Web server 887
- Web server plug-in
  - MaxConnections 861
- Windows 886–887
- Windows 2MSL 887
- Windows MaxUserPort 886–887
- Windows TcpTimedWaitDelay 886–887
- Windows ThreadsPerChild 892
- Windows TIME\_WAIT state 887
- XML parser selection 905
- Performance tuning parameter hotlist 844
- Persistence
  - Database 59
- Persistent message 644
- Persistent session management 178
  - Configuration 285
  - Database session persistence 285
  - Memory-to-memory session replication 285
- Persistent sessions 23, 476
- Pessimistic read 815
- Pessimistic update 815
- Pessimistic update - Weakest Lock At Load 815
- Phantom reads 866
- Ping 110
- PingServletToMDBQueue 593–594, 596
- Plug-in
  - See ORB plug-in
  - See Web server plug-in
- Workload management
  - Workload management
    - Plug-in 185
- Plug-in configuration
  - ClusterAddress 156
  - ServerCluster tag 156
- Plug-in configuration file
  - Generation 151
  - Settings 151
- Plug-in refresh interval 151
- Plug-in workload management 14
- plugin-cfg.xml 80, 151, 295, 572
- PMI 645, 683, 686–687, 701, 705, 717, 723, 725, 745, 836
  - Counters 687
  - MBeans 689
  - Metrics 689
  - Request Metrics 687
  - Request metrics 686
- PMI client 687, 689–690, 696
- PMI Client API 688
- PMI Collector 688
- PMI data 692
- PMI instrumentation 686, 723
- PMI request metrics 725–726
  - Enabling using Administrative Console 725
  - Level
    - DEBUG 725
    - HOPS 725
    - NONE 725
    - PERF\_DEBUG 725
- PMI service 689, 697, 700, 705, 716, 841
  - Enabling using Administrative Console 697
  - Enabling using wsadmin 698
  - List objects 698
- Point-to-point 608, 633, 640–641, 657, 666
- Point-to-point messages 605
- Point-to-point messaging 418, 423, 668
- Pool Size 852
- POP3 110
- Port 138
  - Unique port number 139
- Positive weights 158
- Precompilation 853
- Prefer local 233, 243, 281
  - Configuration 237
- prep 908
- Prepared statement 853
  - Cache size 853

- Prepared statement cache 745
- PrepStmt Cache Discard 854
- Primary cluster 249
- Primary servers 346, 349
- PrimaryServers tag 159, 163, 349
- Private network 437
- Process affinity 230, 233, 240, 243, 246
- Process high availability 316, 523
- Process ID 767
- Process isolation 68
- Processing
  - Up-front 870
- Processing bottlenecks 847
- Production environment 136
- Profiler 759–760
  - Graphical views 761
  - Analyzing execution sequences 761
  - Analyzing garbage collection behavior 761
  - Analyzing object creation 761
  - Analyzing object references 761
  - Analyzing the execution of an application 788
  - Analyzing thread interaction 761
  - Class Instance Statistics 780
  - Class Method Statistics 781
  - Client 763
  - Column Information 777
  - Configuration 764
  - Configuration settings 774
  - Data collection 770
  - Display information 777
  - Execution Flow view 786
  - Host Interaction View 790
  - Host process 762
  - Instance Statistics 783
  - Java Profiling Agent 762
  - Method Invocation Table 790
  - Method Invocation View 789
  - Method Statistics 789
  - Object and Class Interaction (Sequence Diagram) 788
  - Object References Table 784
  - Output 768
  - Package statistics 779
  - Process Interaction View 791
  - Profiling data 761
  - Rules 769
  - Sequence diagrams 761
  - Statistical profiling views 761
  - Testing methodology 775
  - Thread Interactions View 792
  - Using 775
  - Views 776
- Profiler agent 701
- Profiling
  - Class methods 782
  - Filters 769–770
  - Instances of classes 783
  - Method calls 786
  - Methods 781
  - Program execution 788
- Profiling and Logging Perspective 766
- Profiling mode 764
- Profiling Monitor view 775
- Profiling Perspective 763, 775
- Profiling project 768
- Profiling tools 759–760, 804, 808
- Project cycle 760
- Properties file 226
- Provider
  - JDBC 852
- Provider URL 224
- pthread\_mutex 588
- pub/sub 641, 653, 658
- Public network 437
- Publish and subscribe 608
- Publish/subscribe 36, 640, 644, 658, 665
- Publish/subscribe broker 620, 661
- Publish/subscribe messaging 418, 424, 617, 638, 668
- pvccreate 452

**Q**

- QCF 598, 601, 647
  - Maximum connection pool size 628
  - Minimum connection pool size 628
- QCF See Queue connection factory
- Queue 848, 851
  - Local 656
- Queue connection factory 593, 595
- Queue definition 647
- Queue destination 593, 595, 598, 636
- Queue destination settings 615
- Queue manager 424, 593, 595, 614
  - Dead Letter Queue 634
  - Failure 425
  - Local 654

- Name 429
- Receiving 657
- Remote 653
- Sending 657
- Queue size 847–848
- QueueConnection 595, 598, 604, 607
  - Cache 600
- QueueConnection pool 682
- QueueConnections 626
  - Number of 610
- QueueReceiver 635
- QueueSender 615
- QueueSession 595, 598, 604, 607, 624, 626, 682
- Queuing before WebSphere 848
- Queuing network 846, 848

## R

- RAC See Real Application Cluster
- RAID 379, 410, 432, 441, 906
- RAID array 845
- Random 156, 159
- Range statistic 690, 696
- Ratio calculation 847
- Rational Purify 808
- Rational Robot 775, 805
- Rational TestStudio 823
- Raw device 484
- rcp 484
- Read committed 815, 866–867
  - Oracle performance 866
- read only method 814
- Read uncommitted 815, 866–867
- Read/write ratio 901
- readObject() 809
- read-only method 814
- Real Application Cluster 317
- Recover a failed node 924
- Recovery time 336
- RECOVERY.RETRY.INTERVAL 636
- Red Hat 588
- Red Hat Advanced Server 2.1 884
- Red Hat Linux Advanced Server 2.1 588
- Redbooks Web site 945
  - Contact us xxiii
- reduce memory usage 865
- Redundant data 321
- Reference
  - Static 808

- reference LookupTime 696
- Reflection 817
- Refresh interval
  - Web server plug-in 152
- RefreshInterval 888
- Regenerate plugin-cfg.xml 539
- Reload enabled 864, 870
- Reload interval 864, 870, 888
- reloadEnabled
  - Web module 152
- reloadInterval
  - Web module 152
- Remote Method Invocation 64, 711
- Remote queue manager 653
- RemoteException 474
- reorgchk 908
- Repeatable read 815, 866
  - DB2 866
- Repetitive test
  - Memory leak 875
  - Module level 875
  - System level 875
- Replication domain 179, 281, 283, 286, 543, 556
- Replication Entry 282
- Replication entry 283
- Replicator 283
- Replicator entry 179, 286, 543, 556
- Request filtering 895
- Request Metrics 687, 723
  - Filtering mechanism 724
  - Filters
    - EJB method name 724
    - Originator IP 724
    - URI 724
  - Trace record format 726
    - Correlators 726
  - Trace records 726
- Request metrics 686
- Required 814, 868–869
- Requires New 814
- RequiresNew 868–869
- resolv.conf 452
- resource allocation 870
- Resource Analyzer 705
- Resource constraint problems 724
- Resource group 437
  - Cascading 437
  - Concurrent access 437
  - Rotating 437

- Resource leaks 843
- Resource lock 625
- Resource references 594
- Response filtering 895
- Response time 35, 693
- Restore
  - Consideration for system registry 921
  - Network Deployment configuration 915
  - Plug-in configuration file 924
  - Property files 924
  - Repository synchronization 926
  - SSL keyring files 924
  - Using restoreConfig
    - Re-install fixes 923
- restoreConfig 915, 917, 923, 925
- Retry interval 153
- RetryInterval 162, 209–210, 888, 895
  - High availability
    - RetryInterval 352
- RetryItException 472
- Return address 123
- Reverse proxy 65, 129
- RFC 2616 531
- RFC 3143 531
- RISC System Cluster Technology 436
- RMI 390, 726, 855
- RMI over IIOP 689
- RMI *See* Remote Method Invocation
- RMI/IIOP 813, 855
- Rollback 638
- Rotating resource group 437
- Round robin 240–241
  - Server weights 156
  - Turn off 156
  - With weighting 156
- Route 104
- Route tag 150
- Routing algorithm 370
- Routing algorithm overrides
  - Affinity 370
  - Applied (weak) affinity 370
  - In-process 370
  - Prefer local 370
  - Transactional (strong) affinity 370
- RqThrottle 899
- RS232 serial line 379, 410, 441
- runmqsc 665
- runstats 909
- Runtime delays 870

- Runtime Performance Advisor 746, 838
  - Advice Configuration 748
  - Advice configuration 751
  - Configuration 748
  - Output 751
  - Using 749

## S

- SAM 454
- Sample topology 78, 256
- Saturation point 840–841
  - Maximum concurrency point 850
- saturation point 849
- scadmin 462
- Scalability 4–5, 18, 56, 94
  - Horizontal and vertical combined 19
  - Horizontal scaling 19
  - Vertical scaling 18
- Scalable data service 319
- Scaling
  - Vertical 808
- Scaling techniques 33, 44
  - Aggregating user data 47
  - Appliance server 46
  - Batch requests 47
  - Caching 48
  - Creating a cluster of machines 46
  - Managing connections 48
  - Segmenting workload 46
  - Using a faster machine 45
- Scope 611, 619, 646
  - Cell 611
  - Node 611, 646
  - Server 611
- SCS 407, 421
- Security 4, 9, 63
  - Cluster member 20
  - Performance 9
  - SSL communication 9
- Security cache timeout 9, 901
- Segue SilkPerformer 823, 835
- SELECT 815
- Selector 630, 632–633
- Separator 167
- Sequence diagram 761
  - Class Interaction 761
  - Host Interactions 761
  - Object Interactions 761

- Process Interactions 761
- Thread Interactions 761
- Serializable 815, 866
  - Dirty reads 866
  - Nonrepeatable reads 866
  - Phantom reads 866
- Serialization 246
- Serializing 813
- Server
  - Performance data 690
- Server address 123
- Server affinity 27
  - Active cookie 120
  - Enterprise JavaBean 27
  - Entity bean 27
  - HTTP session 27
  - Load Balancer 118
  - Passive cookie 119
  - SSL session ID 120
  - Stateful session bean 27
  - Sticky to source address 119
- Server certificate 143
- Server cluster 12, 134
  - Workload management 17
- Server selection policy 232, 240, 243
- Server tag 150–151
- Server template 346
- Server weighted round robin routing
  - Configuration 234
- Server weights 15, 156–157, 232
  - maxWeight 158
- ServerCluster tag 150, 156, 162
- ServerLimit 898
- Service group 463
- service() 812, 900
- Servlet 2.3 specification 167
- Servlet clustering 14
- Servlet redirector 65
- Servlet response time 707
- Servlet session manager 694
- Servlet workload management 14
- Servlet/JSP result cache 550, 583
- Servlet/JSP result caching 544, 546, 583
  - Configuration 546
  - Demonstration 547
- Session
  - In-memory 808
  - Memory-to-memory replication 809
  - Object 809, 812
  - Reference 809
  - Storage of data 810
- Session affinity 72, 159, 166–167, 361, 819
- Session cache 745
- Session clustering 23
- Session ID 167
- Session identifier 167, 170
  - Clone ID 178–179
  - cookies 170
  - SSL ID 170, 172
  - URL rewriting 171
- Session management 4, 21–22, 166
  - Behavior 188
  - Configuration levels 170
  - Create database 175
  - Database persistence 59
  - DRS 23
  - Enterprise JavaBean 25
  - Garbage collection 809
  - Memory-to-memory replication 23, 59, 178
  - Persistent sessions 23, 59, 178
  - Replication domain 179
  - Replicator entry 179
  - Session clustering 23
- Session manager
  - Levels 362
    - Application server Web container level 362
    - Enterprise application level 362
    - Web module level 362
- Session persistence 59, 168
  - Database persistence 168
  - Failover 173
  - In-memory 364
  - Memory-to-memory replication 168, 364
  - Persisted to database 364
- Session persistence database 809
- Session pool 591, 610, 647, 682
  - Maximum sessions 600, 602
- Session pool maximum 610
- Session pools 679
- Session state 8
- Session support
  - Persistence and failover 361
  - Remove 360
  - Session data loss 362
  - Session update methods 362
- Session time-out 173
- SESSIONMANAGEMENTAFFINI 173
- SessionSampleURLRewrite 930

- setAutoCommit 470
- setCorrelationID method 632
- setDeliveryMode() 615
- setInstrumentationLevel 701
- Shared disk array 432
- Shared resources 810
- Shutdown 456
- Sikatra JProbe 808
- Simultaneous message processing 621
- Single machine topology 60
- Single point of failure 61, 326, 613, 643, 653
  - Eliminating all 523
- Single threaded 808
- SingleThreadModel 812
- Singleton object 807
- Site Selector 98
- smit 378, 409, 440, 884
- smitty 884
- snoop servlet 138, 184
- SOAP 259, 302, 390, 689
- Socket states 842
- Software
  - Requirements 256
- Solaris kernel semsys
  - seminfo\_semopm 886
- Spawned threads 818
- Spawning 818
- Speedera 529
- SPOF 61, 326, 420, 429, 613, 643, 653
  - Database 178
  - Eliminate 438
- SQL 853
- SQL Server 466
- SQL statement 815
- SQLException 468
- SSL 110, 568
  - Advisor 124
  - Bulk encryption/decryption 902
  - Cipher suite 903
  - Connections 902
  - Handshake 120, 902
  - Hardware accelerator 903
  - Performance 902
  - Session time-out 173
- SSL ID 170, 172, 191
- SSL keyring files 920
- SSL session ID affinity 120
  - Configuration 126
- Stack traces 818
- StaleConnectionException 469, 476
- startManager 922, 925
- startNode 923
- startServer 699
- StartServers 891, 898
- State information 166
- Stateful interactions 22
- Stateful session bean 25, 218, 815
  - Server affinity 27
- Stateful session bean home objects
  - clustering of 218
- Stateless interactions 22
- Stateless session bean 25, 217, 813
- Statement Cache Size 854
- Static content 62, 528
- Static field 808
- Static reference 808
- Static SQL statements 908
- Static variable 807
- Statistical profiling view 761
  - Class statistics 761
  - Instance statistics 761
  - Method statistics 761
  - Package statistics 761
- stats\_time 909
- stdout 812
- Steady memory utilization 877
- Sticky to source address 119
- stopManager 921, 924
- stopServer 699
- Stored procedures 817
- Stream 146, 356
- stress See WPT
- String concatenation 817
- Strings 817
  - Manipulation operation 817
- stty 379, 410, 441
- Submodule
  - Performance data 690
- Subscriber queue name 429
- Subscription
  - Durable 654, 666
  - Non-durable 642, 654, 666
- Sun Cluster 459
  - Cluster verification 463
- Sun Cluster Solstice DiskSuite 461
- Sun JDK 1.3 HotSpot
  - new generation pool size 880
  - server warm-up 880

- Sun ONE Web Server 10, 898
- SunPlex 460
- Supports 814, 868–869
- Surrogate-Capabilities 571
- SUSE Linux Enterprise Server 8 SP2A 885
- Swapping 881, 891
- swinstall 452
- Symptom database *See* Log Analyzer
- Synchronization 810
  - java.util.Hashtable 811
  - java.util.Vector 811
  - Method 811
  - Object 811
  - Overhead 811
- Synchronization points 875
- syncNode 922
- Synthetic transaction 724
- System failover 322
- System level metrics 694
- System test
  - Memory leak 875
- System.arraycopy() 818
- SYSTEM.CLUSTER.TRANSMIT.QUEUE 426
- System.gc() 871, 875
- System.out 687
- System.out.println() 811–812
- SystemOut.log 650, 678, 726, 746

## T

- TAF *See* Transparent Application Failover
- Target queue 595
- TaskGuide 379, 410, 441
- TCF 600–601, 605, 619, 647
  - Client ID 668
  - Maximum connection pool size 628
  - Minimum connection pool size 628
- TCP initial connect timeout 889
- TCP initial connect timeout 889
- TCP sockets 907
- TCP stack 889
- tcp\_keepinit 890
- Technology preview
  - Performance Advisors 745
- Telnet 110
- Test applications
  - BeenThere 290
  - Trade3.1 299
- TestMaker 835

- TestNetwork 835
- Thick servlet redirector 65
- Thin servlet redirector 65
- Thread 693, 702, 853, 855, 857, 862, 891
  - Web server 862
- Thread deadlock 734
- Thread dump 734
- Thread pool 694, 707, 851, 855–857, 862
  - Web container 856–857, 863
- Thread safe 608
- Thread starts 701
- Thread synchronization 734
- Thread waits 693, 702
- ThreadAnalyzer 734
  - Deadlock detection 741
  - Deadlock Detection Report 741
  - Project 735
  - Thread dump 734
  - Thread dumps 736
  - Usage example 743
  - Views 740
- ThreadAnalyzer
  - Obtain thread dump 737
- ThreadLimit 897
- Threads 599, 694
  - Idle 898
  - Maximum number 897
  - Spawned 818
- Threads of work 597
- ThreadsPerChild 891–892, 897
- Threshold 806
  - com.ibm.ejs.wlm.MaxCommFailures 372
- Throughput 5, 69, 839
- Throughput curve 849
- Time statistic 690, 696
- TIME\_WAIT 858–859
- TimesTen 531
- Time-to-live 530, 590
- Tivoli Performance Advisor 837
- Tivoli Performance Viewer 622, 628, 651, 679–681, 683, 686–687, 691, 697, 700–701, 703, 705, 707, 711–713, 719, 721, 746, 757, 836, 841, 851–852, 854–855, 857, 873, 876
  - Change buffer size 722
  - Change scale of a counter 720
  - Connect to iSeries 709
  - Connector ports 708
  - Data view 721
  - JCA connection pools

- Monitors 681
- JMS statistics 680
- New performance data 717
- Record a log file 719
- Recording data 719
- Refresh data 717
- Replay a log file 719
- Resource Selection tree 717
- Security enabled 711
- Setting instrumentation levels 712
- Specify refresh rate 721
- Starting 708
- Stopping 711
- Tables and views 717
- Viewing JVMPi output 704
- Tivoli Performance Viewer client 706
- Topic Connection Factory *See* TCF
- Topic destination 647
- Topic destination settings 615
- TopicPublisher 615
- TopicSubscriber 618
- Topology 53
  - Single machine 60
- Topology selection 88
  - Best possible? 86
- toString() 807
- Total memory usage 877
- Tivoli Performance Viewer
  - See also* TPV
- TPV
  - Infrastructure 757
- TPV Advisor 746, 754
  - Configuration 753
  - Enable data collection 754
  - Output 757
  - Replay performance data log file 755
  - Save data in log file 755
- Trace 733
  - GenPluginCfg 184
  - Web server plug-in 183
- TraceFormat 733
- Trade3 427, 439, 533, 547, 552, 592–593, 640, 760
  - Benchmark 585
    - Using command and EJB result caching 585
    - Using ESI 585
    - Using servlet and JSP result caching 585
    - Without caching 585
  - Cluster installation script 306
  - Installation 302, 306, 649
  - JMS functionality 641
  - Performance
    - Using servlet and command caching 588
    - Using servlet, command, and ESI caching 588
    - Without caching 588
  - Trade3 home servlet 547
  - Trade3.1 298
    - Populate database 309
  - TradeBrokerMDB 596
  - Transaction 25, 246, 905
  - Transaction affinity 28, 218, 233, 240, 243, 246
  - Transaction flow 724
  - Transaction isolation level
    - Read committed 815, 866
    - Read uncommitted 815, 866
    - Repeatable read 815, 866
    - Serializable 815, 866
  - Transaction isolation levels 815
  - Transaction log 905–906, 909
  - Transaction management 814
  - Transaction manager 694
  - Transaction rollback 604, 906
  - Transaction server 33
  - Transaction support 614, 616
  - Transaction type 814
    - Mandatory 814
    - NotSupported 814
    - Required 814
    - Requires New 814
    - Supports 814
  - Transactional (strong) affinity 370
  - Transactions per second (tps) ratio 847
  - TRANSIENT 248
  - TRANSIENT exception 246
  - Transparent Application Failover 317, 486
  - Transport 138
    - Setting up multiple transports 141
  - Transport protocol 135
  - Transport tag 150
  - Transports 138
    - Setting up 140
    - Settings 139
  - try {} catch{} block 818
  - TTL 517
  - Tuning 50
  - Tuning failover 207
  - Two phase commit 617



Two-level failover 349

## U

UDDI 259  
ulimit 883, 885  
UML *See* Unified Modelling Language  
Unbounded ORB Service Thread Pool Advice 748  
Uncoordinated processes 340  
Unified Modelling Language 788  
UNIX 66  
UPDATE/UPDATE pair 815  
Updateinstaller 925–926  
Up-front processing 870  
Upstream reply wait 734  
Uptime 522  
Uri tag 149  
UriGroup tag 149  
URL encoding 171, 194  
URL rewriting 171, 194  
    Example 195  
    Session identifier  
        URL rewriting 170  
    urtest Web module 932  
User to business 38–39  
User to data 36  
User to online buying 37

## V

valueUnBound() 810  
Variable  
    Final 807  
    Scope 807  
    Static 807  
VCS 407, 421  
VCS *See* VERITAS Cluster Server  
Vector 876  
-verbosegc 878  
VERITAS Cluster Server 463  
VERITAS Cluster Volume Manager 461  
Versant enJin 531  
Vertical scaling 18, 67, 808  
vgcreate 452  
vgextend 452  
Virtual host 136, 378, 534, 675  
    admin\_host 137  
    default\_host 137  
    PluginVirtualHost 137  
    Servlet request 136

Wildcard 138  
Wildcard settings 137  
Virtual host name 378, 408  
Virtual page size 886  
VirtualHost tag 149  
VirtualHostGroup tag 141, 149  
vmstat 877, 910

## W

WaitingThreads count 899  
WAR file *See* Web archive file  
Web application  
    Execution 762  
Web application performance 792  
Web archive file 62  
Web client 12  
Web container 134, 221, 257, 594, 856, 858  
    Availability 80  
    Clustering and failover 347  
    Failure 197, 348  
    Failure during request 204  
    Failure with active sessions 198  
    HTTP transport 140–141  
    HTTPS transport 140–141  
    Port 135  
    Requests 847  
    Setting up 136  
    Setup 135  
    Transport 138  
Web container threads 599  
Web container workload management 256  
Web module  
    Auto-reload 151–152  
    Disabling auto-reload 153  
    reloadEnabled 152  
    reloadInterval 152  
Web performance analysis 838  
Web Performance Tools 356, 823  
Web primitive 650, 678  
Web server 10, 33, 61, 570  
    Availability 80, 94  
    High availability 498  
    Maximum concurrency thread setting 862  
    Performance 94  
    Process-based 894  
    Reload interval 888  
    Remote from application server 61  
    Scalability 94

- Thread-based 894
- Threads 50
- Workload management 14
- Web server access log 212
- Web server caching 256
- Web server cluster
  - Creation 267
- Web server error log 180, 183
- Web server plug-in 10, 139, 346, 530, 570, 724, 856, 858–859, 890
  - Config element 152
  - Configuration file 147
  - Configuration file generation 151
    - Manual 153
  - Disabling workload management 114
  - Failover 161, 163
  - Failover tuning 207
  - GenPluginCfg trace 184
  - Log file 180
  - Log location 181
  - Logging 180
  - LogLevel 180
    - Stats 181
  - Marking down cluster member 161
  - Normal operation 184
  - Primary and backup servers 163
  - Processing requests 144
  - Refresh interval 152
  - refreshInterval attribute 152
  - Regenerate from Administrative Console 154
  - Regenerate using GenPluginCfg command 155
  - Regeneration 294, 306
  - Retry interval 162
  - RetryInterval 162
  - Suppress load balancing 159
  - Trace 183
  - Troubleshooting 180
  - Workload management 14, 144, 341
  - Workload management policies 156
- Web server plug-in caching 533
- Web services 28, 259, 695
  - SOAP 259
  - UDDI 259
  - WSDL 259
- Web services gateway 694
- Web site classification 36, 42
- Web site performance
  - Caching 798
  - Downloads of large objects 798
  - Number of embedded objects 798
  - SSL 798
- WEB-INF 547
- WebLOAD 835
- WebSphere
  - Administrative Console 11
  - Administrative domain 74
  - Administrative service 11, 689
  - Application server 10
  - Availability levels 328
  - Basic configuration 4
  - Cell 74
  - Cell configuration 277
  - Cluster 16, 76, 143, 276
  - Cluster creation 276
  - Configuration repository 11
  - Database instances 493
  - Deployment Manager 6, 10
  - EJS WLM 20
  - Embedded HTTP transport 11
  - High availability 317
  - InfoCenter 150, 844
  - Java Message Service (JMS) Server 11
  - Load Balancer 83, 96
  - Load Balancer custom advisors 83
  - Network Dispatcher custom advisor
    - Sample 113
  - Node Agent 10
  - Plug-in 134
  - Plug-in WLM 14
  - plugin-cfg.xml 295
  - Resource analyzer 207
  - Runtime Messages 746
  - Single Server 216
  - Thread pool 745
  - Topology 53
  - Trade3.1 298
  - Web server plug-in 10
- WebSphere Application Server
  - Application development 804
- WebSphere Application Server topology 623
- WebSphere Application Server V5.1 256
- WebSphere Business Integration Event Broker
  - 621, 651, 655, 659
  - Configuration 661
    - Configuration manager 661
    - Message broker 661
    - Message brokers toolkit 661
- WebSphere Business Integration Event broker 591

- WebSphere Business Integration products 620
- WebSphere Caching Proxy 533
- WebSphere clusters 76
- WebSphere Command Framework API 551
- WebSphere Commerce 530
- WebSphere configuration
  - Dynamic 8
  - Mixed 8
  - plugin-cfg.xml 114
- WebSphere Dynamic Cache 819
  - Troubleshooting 564
- WebSphere Dynamic Cache service 564
  - Troubleshooting 564
- WebSphere dynamic caching 533
  - Command result caching 533
  - JSP resulting caching 533
  - Servlets 533
- WebSphere Edge Components 95, 260, 530
- WebSphere Edge Server 583
- WebSphere Embedded JMS server
  - Functionality 424
- WebSphere external caching 533
  - Using Caching Proxy 533
  - Using ESI 533
  - Using FRCA 533
- WebSphere Failover Unit 381, 411
- WebSphere for z/OS 368
- WebSphere InfoCenter 150, 844
- WebSphere internal messaging 283
- WebSphere JMS provider 611, 647
- WebSphere MQ 591, 593, 595, 611, 615, 642, 651
  - Backout queues 638
  - Backout threshold 638
  - Cluster 655–656
  - Cluster configuration 425
  - Clustered queues 428
  - Clustering 424
  - Clustering workload management 651
  - Configuration 655
  - Create cluster 659
  - Create queue 660–661
  - Create queue manager 657, 659
  - Dead Letter Queue 657
  - Failover and workload management 653
  - Messaging for heterogenous environmen 424
  - Repository queue managers 428
  - Stuck messages 428
  - Workload management 656
- WebSphere MQ 5.3.0.1 423
- WebSphere MQ clustering 651
- WebSphere MQ Event Broker 2.1 424
- WebSphere MQ Explorer 426, 657, 659, 678
- WebSphere MQ infrastructure 653
- WebSphere MQ JMS provider 418, 611, 624
- WebSphere MQ queue manager 630
- WebSphere MQ Real-Time transport 620
- WebSphere MQ server 593, 626
- WebSphere on iSeries 709
- WebSphere Queue Connection Factory
  - Configuration for Trade3 647
- WebSphere Runtime Messages 752
- WebSphere runtime resources 707
- WebSphere Studio Application Developer
  - Performance measuring 760
  - Performance profiling 760
- WebSphere Topic Connection Factory
  - Configuration for Trade3 648
- WebSphere WebSphere Business Integration Event Broker 653
- webspheredb2.config 453
- websphereoracle.config 453
- WebStone 835
- Weight
  - Current 158
  - Maximum 158
  - Positive 158
- Weighted round robin 156
- WLM *See* Workload management
- Workload characteristics 41–42
- Workload management 4–5, 13, 17, 20, 134, 143–144, 256, 345, 652, 695, 808
  - See also* EJS workload management
  - Backup servers 134–135
  - BackupServers tag 163
  - Browser requests 156
  - Demonstration 259
  - EJB selection policy 21
  - Enterprise Java Services 20
  - HTTPS considerations 143
  - Primary servers 134
  - PrimaryServers tag 163
  - Process flow 147
  - Random 159
  - Server cluster 17
  - Servlet 14
  - Suppress load balancing 159
  - Web server 14

- Web server plug-in 14
- Weighted round robin 135, 156
  - Server weights 157
- Workload management policies 156
  - Random 156
  - Weighted round robin 156
- Workload management selection process 184
- Workload patterns 36, 642
- WPT 356, 823
  - stress 849
- wsadmin 302, 390, 397, 408, 687, 697
  - GenPluginCfg 153, 155
  - list PMIService 698
  - modify 699
  - save 699
  - set 701
  - set perfObjRef 701
  - set sigs 701
  - startServer 699
  - stopServer 699
- WSDL 259

## **X**

- XA coordination support 424
- XA support 614, 616
- XA transactions 625, 906
- XML 540
- XML cache policy file 551
- XML parser 905
- XML tag 150
- Xnoclassgc 882
- XrunpmiJvmpiProfiler 702, 704



**Redbooks**

# **IBM WebSphere V5.1 Performance, Scalability, and High Availability: WebSphere Handbook Series**

(1.5" spine)

1.5" <-> 1.998"

789 <-> 1051 pages







# IBM WebSphere V5.1 Performance, Scalability, and High Availability WebSphere Handbook Series

**Select a starting  
topology for your  
WebSphere Web site**

**Workload manage  
Web server, servlet,  
and EJB requests**

**Explore high  
availability and  
security options**

This IBM Redbook discusses various options for scaling applications based on IBM WebSphere Application Server Network Deployment V5.1. It explores how a basic WebSphere configuration can be extended to provide more computing power by better exploiting the power of each machine and by using multiple machines. It examines a number of techniques:

- ▶ Using the IBM WebSphere Edge Components' Load Balancer to distribute load among multiple Web servers.
- ▶ Using the WebSphere Web server plug-in to distribute the load from one Web server to multiple application servers in a server cluster.
- ▶ Using the WebSphere EJB workload management facility to distribute load at the EJB level.
- ▶ Using dynamic caching techniques to improve the performance of a Web site.
- ▶ Using clustering solutions such as HACMP to meet the high availability needs of critical applications.
- ▶ Using application development best practices to develop a scalable application.
- ▶ Using the performance tuning options available with WebSphere to adjust the application server configuration to the needs of your application.

This book provides step-by-step instructions for implementing a sample, multiple-machine environment. We use this environment to illustrate most of the IBM WebSphere Application Server Network Deployment V5.1 workload management and scalability features.

## INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

### BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)